Gridification and Workflow Scheduling for the German D-Grid

Dissertation

zur Erlangung des Grades eines Doktors
der Naturwissenschaften

vorgelegt von

Dipl.-Inf. Dietmar Sommerfeld
aus Heilbad Heiligenstadt

genehmigt von der
Fakultät für Mathematik/Informatik
und Maschinenbau
der Technischen Universität Clausthal

Tag der mündlichen Prüfung
16. Januar 2013

Vorsitzender der Promotionskomission
Prof. Dr. Jörg P. Müller

Hauptberichterstatter
Prof. Dr. Harald Richter

Berichterstatter
Prof. Dr. Oswald Haan

Berichterstatter
Prof. Dr. Sven Hartmann

Acknowledgment

I want to thank all the people who supported me to make this thesis possible.

At first, I would like to thank my doctoral advisor Prof. Dr. Harald Richter for his instructive guidance and continuous support throughout the past years.

I also want to thank Prof. Dr. Oswald Haan for introducing me to Grid computing and for kindly agreeing to be second reviewer.

I wish to thank Prof. Dr. Sven Hartmann for his considerate acceptance of providing the third review.

I would like to express my sincere thanks to my former project partner Andreas Hoheisel for generously sharing his knowledge on the Generic Workflow Execution Service.

I want to thank all members of the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen for providing a pleasant and inspiring working atmosphere. In particular, I am grateful to Tibor Kálmán, Dr. Ulrich Schwardmann, and Dr. Christian Boehme for the productive discussions and their helpful advice.

Further, I wish to thank all members of the Department of Informatics at Clausthal University of Technology for the pleasant collaboration.

At last, I would like to thank my family and my wife who always supported me with their encouragement and patience.

Abstract

In recent years, scientists from many domains have discovered Grid technology which offers new possibilities for solving problems that are difficult to handle with traditional distributed computing. In the first part of this thesis we examine the process to enable life science applications for execution on the Grid. Such applications often require the analysis of very large data sets and consist of several successive program runs. As a case study we describe the adaptation of the gene-finding tool AUGUSTUS to Grid computing in the context of the MediGRID virtual organization which is part of the German D-Grid. At first we show how the application can be run manually on Grid resources by means of the Grid middleware. Afterwards, we describe how the application is represented as a Petri net workflow comprising several tasks which are automatically distributed to appropriate Grid resources by an advanced workflow engine. Finally, we show how a convenient graphical user interface for end users is created within a portal framework.

In the second part of the thesis we concentrate on general Grid workflow scheduling in MediGRID which is concerned with the mapping of workflow tasks to Grid resources. We start our analysis with measurements of the availability of resources in D-Grid and identify four problems inherent to meta-scheduling that make Grid scheduling nearly always a delicate task. To address these issues we present three methods to estimate the queue waiting times of the prevalent cluster resources. Additionally, we propose three selection algorithms to automatically select the best current estimation method. Evaluation shows that prediction works very well on most resources and recognizes the peaks in waiting times that can last up to hours. For the scheduling of workflow tasks we propose and evaluate a two-tier approach that combines contemporary scheduling strategies for workflows and for Grids, and that is suitable for production environments. The first tier uses a list scheduling heuristic to create a full-ahead schedule of tasks based on predictions of task execution times. The second tier performs a just-in-time mapping of the prioritized tasks to resources according to performance predictions that incorporate the estimated queue waiting times. In the end, the two-tier approach is integrated into the MediGRID workflow engine. We assess the improved scheduling by measurements and demonstrate a significant acceleration of up to 28% in workflow processing compared to the existing strategies.

Contents

Chapter 1

Introduction

The development of distributed computer systems is closely connected to the technological advances in the area of computer processors and computer networks. In the 1980s technological advances accelerated the wide-spread use of distributed computer systems [1]. The development of highly performant micro processors led to computers that were as powerful as former mainframe systems but at a fraction of the cost. At the same time high-speed networks were introduced that allowed rapid data transfer between several computers and at greater distances. As a result it became possible and economical to assemble distributed systems comprising lots of computers with fast innterconnections.

A distributed system is defined as a system that consists of several autonomous computers that communicate via message exchange. It appears to its users as a single coherent system [1, 2]. The involved computers are also denoted as nodes.

The purposes of building distributed systems are similar to those of computer networks. They can serve for function sharing to offer resources with special capabilities, data sharing to make local data available, load sharing to balance load or allow more complex computations, and high-availability by increasing the reliability of the overall system via redundancy. Distributed systems can pursue one or more of these purposes.

A problem associated with distributed systems is the lack of an up-to-date consistent view. This is due to the autonomy of components, parallelism and spatial distance. It means that no node has complete control of cooperative processes, or complete and instantaneous state information which in turn leads to inconsis-

tencies and non-determinism. Many practical problems also result from heterogeneity of resources with respect to architecture, operating system, communication protocols and data representation. In comparison with parallel computers distributed systems make higher demands on security, dynamic software configuration and software complexity [2].

Distributed systems should possess a number of qualities. They should make it easy for users to access the resources and hide to a reasonable degree that resources are physically distributed across a network. Additionally, they should be open by providing services with standardized syntax and semantics. Finally, they should be scalable in size regarding users, resources, geographical expansion, and number of involved organizations [1].

The design and implementation of distributed systems is typically characterized by the principles of distribution transparency and layering. A distributed system is said to be transparent if it is able to present itself to users and applications as a single, abstract and potentially concurrent machine. There are several types of distribution transparency regarding resource access, resource location, resource performance, resource failures, etc. However, full transparency is not possible and not desirable as it can effect performance and comprehensibility. Layering means that the system is structured into levels of abstraction that build on each other and each mask the layer below. The functionality of each layer is subsumed in services and communication between layers occurs via service primitives. Due to the abstractions, layering again adds to transparency [2, 1].

Three definitions of Grid

In the late 1990s Grids emerged as a new kind of distributed systems. The first experience with Grid technology was gained in experiments with gigabit networks such as I-WAY [3]. The I-WAY project established temporary links between 11 high-speed research networks and connected 17 U.S. supercomputer centers to create a first national Grid. In this environment researchers were able to run distributed virtual reality applications on multiple internetworked supercomputers.

The term Grid is derived from comparison with the electric power grid, pointing towards the aim to make computing power as easily accessible as electric

power. This analogy is also expressed in the first definition of Grid by Foster and Kesselman (eds.) in the book "The Grid: Blueprint for a New Computing Infrastructure" [4]:

"A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities."

In this pathbreaking work they explained the motivation of Grid which is the enabling of new classes of applications. After that, they examined the state of the art in existent technologies that provide the essential building blocks to create Computational Grids such as "advanced optical networks, fast microprocessors, parallel computer architectures, communication protocols, distributed software structures, security mechanisms, electronic commerce techniques".

Many of the foundations of todays Grid computing were laid in the subsequent article "The Anatomy of the Grid" [5] by Foster, Kesselman and Tuecke, which also refined the Grid definition. The focus changed towards collaboration and sharing relationships by stating that Grid computing is concerned with "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations." The definition introduces the concept of virtual organizations (VOs) which consist of individuals, institutions and resources that join in a dynamic alliance to work for a common purpose, and collaborate according to the rules that determine the sharing conditions.

VOs can differ vastly in their purpose, duration, scale, type of participants and the resources being shared. Fig. 1.1 shows VOs of different scope that constitute a Department Grid, Enterprise Grid and Global Grid. What they all have in common is that a number of participants "with varying degrees of prior relationship (perhaps none at all) want to share resources in order to perform some task". Furthermore, sharing has a greater extent than simply document exchange. "It can involve direct access to remote software, computers, data, sensors, and other resources." At the same time it occurs "highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs."

Expressing sharing relationships requires mechanisms to formulate policies, to check authentication, and to perform authorization. Because of the dynamicity, sharing relationships can vary over time regarding resources, participants and

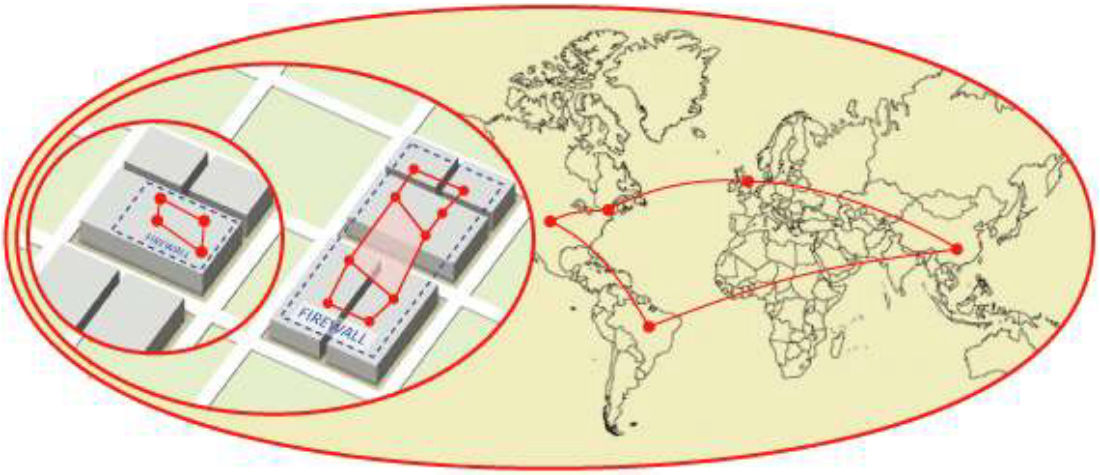

Figure 1.1: Different scopes of Grids: Department Grid, Enterprise Grid and Global Grid (from left to right) [6].

access rights. Furthermore, participants need not be explicitly named, but may be defined in terms of roles. Additionally, dynamicity also implies a requirement for registry mechanisms to discover and describe the conditions that apply at a certain time. Sharing relationships are flexible and not restricted to client-server models, but can extend to peer-to-peer, which means that providers can also be consumers. Therefore, delegation of rights and authority become important.

In [5] the authors conclude: "These characteristics and requirements define what we term a virtual organization, a concept that we believe is becoming fundamental to much of modern computing. VOs enable disparate groups of organizations and/or individuals to share resources in a controlled fashion, so that members may collaborate to achieve a shared goal."

A final definition of Grid was given by Foster in [7]. It can be perceived as a checklist to distinguish Grids from other distributed systems:

"A Grid is a system that coordinates resources that are not subject to centralized control using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service."

This definition is pointed towards large-scale Grids as it emphasizes that users and resources belong to multiple administrative domains which brings up issues of membership, security, policies and accounting. The protocols and interfaces must

be standardized and open to enable interoperability and portability. The quality of Grid services regards their availability, security, capabilities, and applicability. Grids serve complex user demands, at which "the utility of the combined system is significantly greater than that of the sum of its parts."

According to this definition local management systems e.g. for clusters, and application specific systems do not qualify as Grids. Neither do enterprise distributed computing technologies such as CORBA, Enterprise Java or DCOM since they are typically directed at rather static sharing arrangements within single organizations. This meant that most existing technologies were not suitable to establish VOs with the desired range of resources and sharing relationships.

Grid architecture

In the second part of [5] the authors propose a protocol architecture for Grids to enable resource sharing among arbitrary participants. The key issue about this is interoperability which is achieved with common protocols. Standard protocols in turn facilitate the definition of standard services which allow to abstract from specific resource properties. Those can be an obstacle in the development of Grid applications.

The proposed architecture is structured into layers similar to the Open Systems Interconnection model. Components with similar characteristics are grouped in the same layer. Each layer builds on the capabilities of the layer below and and provides capabilities to its upper layer. The Grid architecture which is depicted in Fig. 1.2 follows the "hourglass model". The neck of the hourglass defines a small set of protocols and abstractions that build upon the variety of technologies at the bottom of the hourglass. The top of the hourglass holds the high-level functions which are enabled by the core protocols at the neck.

The fabric layer provides the resources which are shared in the Grid, e.g. computational resources, storage resources, network resources and sensors. Fabric components perform the resource-specific operations and expose their vendor-supplied protocols and interfaces. These may be enriched to support more complex sharing operations.

The connectivity layer defines the communication and security protocols to ex-

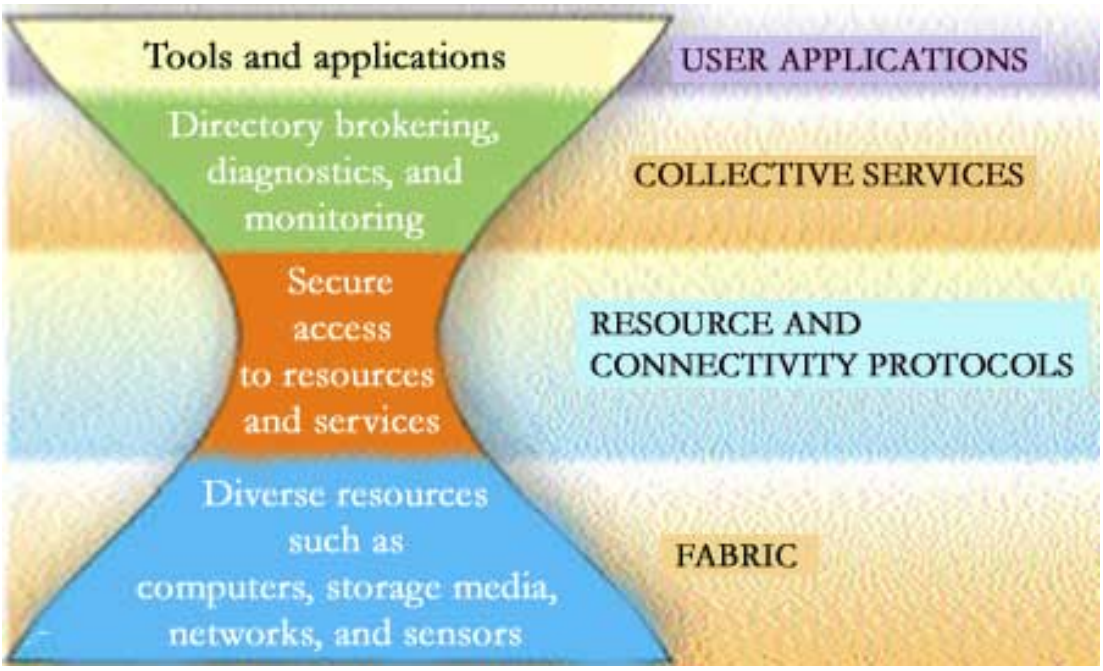

Figure 1.2: Layers in the Grid [8, 9].

change data between fabric resources. The communication protocols are typically taken from the internet protocol suite, most notably IP and TCP. Security comprises aspects like authentication, authorization and encryption which can be achieved for instance by means of TLS.

The resource layer defines protocols that facilitate the initiation, control, monitoring and accounting of sharing operations on individual resources. Its protocols build on fabric layer functions and can be separated mainly into information or management protocols. Together with the connectivity layer, the resource layer forms the neck of the hourglass, therefore its protocols are limited to the fundamental sharing mechanisms. This is necessary because they are widely deployed and must be implemented on top of the variety of different resource types.

The collective layer specifies protocols that are not concerned with single resources anymore but rather coordinate sharing across collections of resources. By resorting to the resource layer functionality they do not place additional requirements on the shared resources. Collective layer protocols enable global and application-specific sharing behaviors at the same time. This means they can range from general-purpose to highly domain specific. The latter may only exist

within single VOs.

At the top of the hourglass resides the application layer which holds the user applications of a VO. The applications employ the resource management, data transfer and discovery protocols defined at the lower layers. Application layer software can include frameworks and libraries with a complex internal structure which are necessary to run the end-user programs.

Grid services

The third among the influential works which shaped todays Grid technology was the article "The Physiology of the Grid" [10] by Foster, Kesselman, Nick and Tuecke. It complements the previous article which was structured according to the protocols in the Grid architecture by focusing on the services that respond to the protocol messages.

With the new perspective all Grid resources are represented as services. The service-oriented view allows the encapsulation of diverse implementations behind a standard interface. The authors denote this as virtualization. It provides distribution transparency in terms of access, location and concurrency. The virtualization facilitates consistent resource access across heterogeneous platforms, mapping of multiple logical resource instances onto the same physical resource, and composition of resources from lower-level resources regardless of service implementation.

The services are defined by means of the Web Services Description Language (WSDL) [11]. WSDL describes the service functionality and defines the service interfaces in a standardized way, therefore all implementations of the service can be invoked in the same way. Additionally, the service interface definition is independent from the service invocation so that multiple optimized protocols can be used for invocation.

By the use of WSDL, the services are aligned with the web services technology. The adoption of web services provides a wide range of benefits, for instance standardized service description and discovery, automated communication stub generation on client and server side, interoperable protocol bindings, and compatibility. However, the web services have to be augmented to provide the complete

functionality required in VOs. To this end the authors propose so-called Grid services, which are potentially transient stateful web service instances. Grid services provide a set of standard interfaces that follow specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, and notification, whereas the conventions address naming and upgradeability. The authors denote the specifications that define the Grid services in alignment with web services as the Open Grid Services Architecture (OGSA).

In summary the view of a Grid is an extensible service-oriented distributed system which provides multiple forms of distribution transparency. Grid services share standardized interfaces and can be aggregated to form higher-order services. The implementation of Grid services can map to native platform functions which facilitates the integration of existing IT infrastructures. OGSA represents an enhancement of web services that improves their applicability for building VOs while only requiring minor extensions to existing technologies.

1.1 Overview

The research presented in this thesis focuses on user-friendly and efficient execution of scientific workflow applications within the virtual organization MediGRID [12] which is part of the German e-Science initiative D-Grid [13]. The VO originates from the initial MediGRID project [14] and follow-up projects such as Services@MediGRID [15]. MediGRID is a community Grid for researchers in the fields of medicine, biomedical informatics, and life sciences. Its end users usually do not have a computer science background, therefore an important aim is to make Grid solutions as user-friendly as possible.

To run applications in MediGRID they have to be adapted to its Grid middleware, a process we denote as gridification. The middleware supports sharing of heterogeneous resources and offers distribution transparency. It is a software layer that is logically placed between the fabric layer with the operating systems and basic communication facilities, and the high-level layer consisting of users and applications [1]. In the Grid architecture, the middleware represents the connectivity, resource and collective layer. It is present on all machines taking part in the Grid and enables a uniform access to the underlying fabric resources

by providing a common set of services onto which the development of distributed applications can be settled.

MediGRID uses the Globus Toolkit 4 (GT4) [16] as its Grid middleware. Globus Toolkit is among the most popular Grid middlewares and has widespread use in Grid projects around the world. In version 4, the architecture of the toolkit was completely changed and implemented with stateful web services that are compliant to the Open Grid Services Architecture [10]. OGSA services are in particular responsible for security infrastructure, execution and data management, monitoring and resource discovery.

On top of the basic GT4 middleware, MediGRID employs further specialized middleware services. The most important one is the Generic Workflow Execution Service (GWES) [17]. GWES is an advanced workflow system for orchestrating the distributed execution of applications on Grid resources. [1] An application workflow [18] consists of several concurrent and sequential program executions as well as intermediate data transfers. This allows to model the dependencies between programs which are also called tasks in workflow terms. At the presentation layer, MediGRID employs a portal framework to provide a graphical interface which is suitable for the end users.

In this thesis we describe the gridification of applications for MediGRID which has to be done by experienced developers. The gridification process involves GT4, the workflow system and the portal framework as required components. It does not require changes to the application source code. We examine the process with AUGUSTUS [19] as an example from the field of bioinformatics. AUGUSTUS is a successful and widely used software to identify gene structures in eukaryotic genomes.

A very important issue to efficiently execute scientific workflow applications is how to optimally schedule the workflow tasks onto the distributed Grid resources, and how to handle inter-task dependencies to minimize waiting time and to maximize the number of runnable tasks. Task scheduling depends on information about the performance parameters that influence the overall completion time of tasks. We identify by tests in the D-Grid that for cluster resources input-queue waiting-times are the key parameter because they can change dramatically within short

[1] Throughout the thesis, we use the term application for complex usage scenarios involving several executables, while the term program stands for a single executable.

time and last up to hours. For this reason we develop three methods to estimate the queue waiting times, and three selection algorithms to automatically select the best estimation method due to the current situation on the resource.

For the scheduling of scientific workflow applications we present and evaluate a novel approach that combines contemporary scheduling strategies for workflow tasks and Grid jobs. It has a manageable scheduling complexity and is robust, i.e. suited for production Grids. The approach consists of two tiers. The first tier operates before the workflow execution starts and creates a full-ahead schedule of the workflow tasks based on predictions of task execution times. For this, we use a list scheduling heuristic similar to the Heterogeneous Earliest-Finish-Time algorithm [20]. In the resulting static schedule, all tasks are assigned ranks which determine static execution priorities. The second tier operates dynamically at runtime and performs a just-in-time mapping of runnable tasks to Grid resources. It supports additional prioritization schemes and selects resources according to performance predictions that incorporate the estimated queue waiting times. The two-tier approach is integrated into the GWES workflow system and assessed by measurements that demonstrate its efficiency and robustness in the D-Grid environment.

1.2 Document Structure

This thesis is separated into two parts. The first part with Chapter 2 describes by means of a case study the adaptation of an application to Grid computing. It starts with a brief introduction of the AUGUSTUS application and its use on a standalone computer. Then, we give an overview of MediGRID's basic Grid middleware Globus Toolkit and demonstrate how AUGUSTUS is run with Globus tools. In the following we describe how the AUGUSTUS usage scenario is represented as a workflow of subsequent program executions and what information has to be provided by application developers to enable automatic distribution of computations in the Grid by the GWES. Afterwards, we demonstrate how a graphical user interface for AUGUSTUS is provided in the MediGRID portal. Finally, we present the results of the gridification and the lessons learned. The chapter closes with a summary discussion.

The second part of the thesis describes our improvements on workflow scheduling

in D-Grid. In Chapter 3 we present the state of the art to allow for distinction from other works in this field of research. We first discuss approaches for the prediction of input-queue waiting-times of compute clusters. Second, we present related works in the domains of Grid scheduling and workflow scheduling in conjunction with their application in workflow systems for Grids. In Chapter 4 we present our measurements of queue waiting times on clusters in the D-Grid and derive consequences for workflow scheduling. Subsequently, we develop our adaptable prediction methodology. We present three alternative methods to predict queue waiting times, three algorithms to dynamically select a method, and the results of the prediction. The chapter concludes with a description of the integration in MediGRID and a summary discussion. Chapter 5 begins with a discussion of the previous workflow scheduling in GWES and its shortcomings. Afterwards, we introduce our new scheduling approach with its workflow-level and Grid-level tiers. Then, we describe the integration of our scheduling approach into the GWES. Finally, we present the results of an assessment of the novel approach in a testbed environment, and judge the approach in a summary discussion. The thesis ends with an overall conclusion in Chapter 6.

1.3 List of own Publications

Parts of the results have been published in reviewed reports, articles and at conferences. These are listed in the following. The gridification of the AUGUSTUS application was a joint effort with T. Lingner from the Department of Bioinformatics at the University of Göttingen. He developed the application portlet.

- D. Sommerfeld, T. Lingner, and H. Richter, "Stepwise Enabling of AUGUSTUS for MediGRID," Tech. Rep. IfI-07-12, Clausthal University of Technology, 2007.
- D. Sommerfeld, T. Lingner, M. Stanke, B. Morgenstern, and H. Richter, "AUGUSTUS at MediGRID: Adaption of a bioinformatics application to grid computing for efficient genome analysis," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 337 – 345, 2009.
- D. Sommerfeld and H. Richter, "A Novel Approach to Workflow Scheduling

in MediGRID," Tech. Rep. IfI-09-07, Clausthal University of Technology, 2009.

- D. Sommerfeld and H. Richter, "A two-tier approach to efficient workflow scheduling in MediGRID," in *Grid-Technologie in Göttingen - Beiträge zum Grid-Ressourcen-Zentrum GoeGrid* (U. Schwardmann, ed.), vol. GWDG-Bericht Nr. 74, pp. 39–51, Göttingen, Germany: Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, 2009.

- D. Sommerfeld and H. Richter, "Problems and Approaches of Workflow Scheduling in MediGRID," in *2009 Fifth IEEE International Conference on e-Science*, (Oxford, United Kingdom), pp. 223–230, 9–11 December 2009.

- D. Sommerfeld and H. Richter, "A Two-Tier Approach to Grid Workflow Scheduling," in *2009 2nd IEEE International Conference on Computer Science and its Applications*, (Jeju, Korea (South)), pp. 267–273, 10–12 December 2009.

- D. Sommerfeld and H. Richter, "Efficient Grid Workflow Scheduling Using a Two-Tier Approach," in *Proceedings of the Ninth HealthGrid conference 2011*, (Bristol, United Kingdom), 27–28 June 2011.

Part I

Gridification

Chapter 2

Case study: Adaptation of a Bioinformatics Application to Grid Computing

2.1 AUGUSTUS

We examine the gridification process by means of a case study with the AUGUSTUS application as an example from the field of bioinformatics. AUGUSTUS [19] is a DNA-sequence-based gene prediction program for eukaryotes, i.e. organisms with a cell nucleus. The purpose of the program is to identify the location and structure of all protein-coding genes in a given genome. Gene prediction is the first and most important step in the analysis of newly sequenced organisms. During the last years, AUGUSTUS has been used in many genome projects, e.g. [28, 29]. A web interface for AUGUSTUS has been available at the website of the Bioinformatics group in Göttingen for several years [30]. Because the computing power that is accessible for the group is limited, the web interface only allows the input of relatively short sequences up to 3 million base pairs. The AUGUSTUS application itself does not have this limitation. By way of comparison, the human genome has about 3 billion base pairs [31].

AUGUSTUS is typical for a whole class of bioinformatic applications: the program uses sophisticated statistical modeling and prediction algorithms which are computationally demanding. It does not require user interaction and can be

run unattended. Input and output data is accessed from a central data storage repository. An application run consists of several phases which are separated by I/O operations that read and write files. Via these files, data exchange between phases is done. The computing phases are loosely coupled, i.e. they have little communication in relation to the computation. AUGUSTUS is distributed as open source software.

AUGUSTUS can be used as an ab initio program, which means that the prediction is based solely on the input DNA sequence. As a second possibility, the software may also incorporate hints on the gene structure coming from extrinsic sources such as BLAST [32] and DIALIGN [33, 34] search results. With external hints, the prediction performance of AUGUSTUS can be increased significantly [35]. On the other hand, the computational costs to find external hints can exceed the costs of the program itself, in particular when large databases are used for BLAST search. Therefore, the search for external hints is not integrated into the AUGUSTUS web interface.

In the following, we will focus on ab initio use of AUGUSTUS. In this case the command line program has two mandatory arguments: the query file and the species. The query file contains the input DNA sequences in uncompressed (multiple) FASTA [36] format. The species parameter denotes where the DNA sequences originate from, e.g. human, fruit fly, baker's yeast, etc. There are several optional parameters which influence the way the genes are predicted. The output of AUGUSTUS is written to the command line output or into user-specified files. The output is compatible to the General Feature Format which can be visually interpreted by means of a genome browser [37]. A typical command line looks like:

```
augustus --species=human example.fa > outputfile
```

Running AUGUSTUS on a single computer is relatively easy. However, depending on the number and length of sequences and the parameter values, the gene prediction can be computationally very intensive. For example, the application to the human genome with its 3 billion base pairs can take up to several weeks. A possibility to speed up the computation without modifying the program source code is to parallelize the computation. This can be done by splitting the input file

into smaller files which contain fewer sequences, and to run the program on these input files separately. Long sequences can be split into several smaller sequences if a suitable gene sequence overlap according to maximum gene size is used. The possibility to split the input data holds also for the search for external hints.

Data parallelization provides the possibility to distribute the computation across several machines, for example across the nodes of a local cluster. Typically, these nodes have a shared file system, so the handling of input and output data is easy. However, additional effort is necessary for handling the resource management system and for checking that all submitted jobs are executed properly.

The organizational effort increases if local cluster resources are not sufficient for the demands of the computation, e.g. when external hint search is incorporated. In this case, further suitable resource providers have to be found to support the computation. For the AUGUSTUS application, this means that the input data must be split and distributed to the sites where the program is intended to be run. Without Grid technology, this necessitates to deal with several potentially different authentication and resource management systems. In the end, the output files have to be transferred back to the user for result analysis.

The benefit of the parallelization is the decrease in overall execution time, allowing to tackle larger problem sizes which cannot be addressed by the own local resources. Fortunately, some of the organizational problems such as handling of user accounts and job submissions on multiple sites can be solved by the middleware layer of the Grid.

2.2 Executing AUGUSTUS on the Grid

MediGRID was set up as a service Grid that uses the Globus Toolkit as its basic Grid middleware. The Globus Toolkit is an open-source set of services and software libraries that support Grids and Grid applications. It addresses issues of security, execution management, data management, information discovery, communication, fault detection, and portability. Since version 4 the toolkit is implemented with stateful web services that follow the specifications of the Web Services Resource Framework (WSRF) [38]. In the same time, GT4 is a realization of the Open Grid Services Architecture which defines a common and

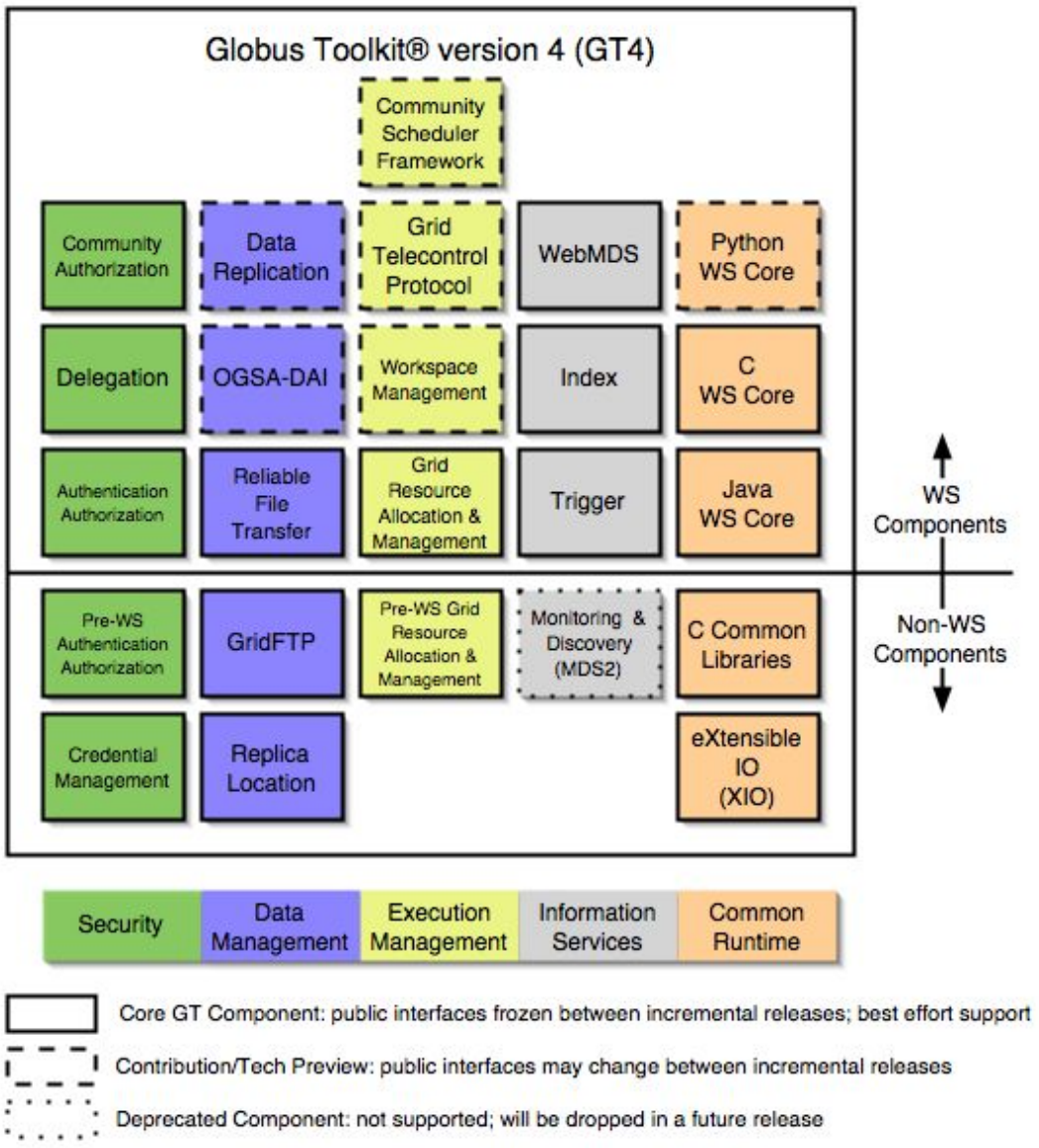

Figure 2.1: Components in the Globus Toolkit [39].

standard architecture for Grids. The goal of OGSA is to standardize practically all services that are typically required in a Grid by specifying a set of standard interfaces for these services.

Even though most Globus services are implemented on top of WSRF, the toolkit also includes some components that are not implemented with stateful web services. These are called the non-WS components and remained from earlier versions of the toolkit. Fig. 2.1 shows all services and software libraries that are

part of the GT4 distribution.

The first step before we can execute AUGUSTUS on the Grid is to provide for authentication and authorization. The Grid security infrastructure (GSI) of GT4 is based on a public key infrastructure. This means, every user and every server in the Grid is authenticated by a pair of public and private key. The public keys are signed by trusted certificate authorities (CA) and are subsequently used as X.509 [40] identity certificates. Thus, the first step to use Grid resources, is to get a user certificate from a CA that is recognized by the Grid's security policy. The pair of user certificate and private key is called Grid credential. To prevent the private key from leaving its secure environment on the user's computer, it is used to sign the public key of a new pair of short-lived proxy credentials. These proxy credentials are then used to authenticate the user in the Grid.

To get an authorization for the machines in the Grid the user has to become a member of the Grid's Virtual Organization (VO). There are several solutions for VO membership registration. MediGRID uses the VOMRS [41] solution. The authorization is achieved with so-called grid-mapfiles that are present on every machine. For every Grid user, they contain one line which states the distinguished name from the user's certificate and a local Unix account. The distinguished name is a unique identifier represented by a string that consists of attribute-value pairs. When a Grid user accesses a server with his proxy credentials, first the authenticity of the proxy certificate is checked, and afterwards the server looks for the distinguished name in the grid-mapfile and maps the user onto the respective local Unix account. The following example maps the Grid user "Dietmar Sommerfeld" on the account "dgmd0010": [1]

```
"/C=DE/O=GridGermany/OU=Gesellschaft fuer wissenschaftliche Datenverar\
beitung mbH/CN=Dietmar Sommerfeld" dgmd0010
```

Thereafter, all user actions are performed under that account and with its specific permissions. Besides authorization and authentication, the GSI provides secure communication via TLS [42] and enables single sign-on, which means that a user only has to log-on once at one machine with his proxy credentials. Afterwards,

[1] Note that we use the backslash character \ to denote a line break because of limitations in text width.

the user can use services on any server in the Grid during the validity period of his proxy credentials (usually twelve hours).

To find suitable machines for the execution of the AUGUSTUS application, a resource discovery has to be performed. For this purpose, each GT4 installation provides the Monitoring and Discovery Service (MDS). This service collects basic resource information about the hosts such as processor architecture, operating system, available memory, and provided services in XML format. Each local MDS is typically configured to upstream its information to a server which aggregates the data of several machines. This way, a hierarchical structure is formed with local MDS instances at the bottom and a so-called central index server at the top. In MediGRID, the central index server provides the WebMDS front-end which can be accessed with a web browser. For this, WebMDS uses XSL transformations [43] to convert the XML data of MDS into HTML pages.

Normally, the specific application is not installed initially on the Grid machines and therefore has to be deployed before its first use. There are several approaches for application deployment in the Grid. First of all, the application developers need interactive access to every kind of machine the application shall be executed on. This is necessary for compilation and for testing in the respective runtime environment. For this purpose, computing centers in D-GRID provide so-called interactive nodes that have the same software installation as the actual cluster nodes which do not allow user login. The standard deployment approach is to have a common software stack on all machines in the Grid. In this case, the applications are pre-installed by the system administrators and application developers. Another possibility is to do a dynamic deployment at runtime by transferring already compiled executables from a software repository to the execution hosts. MediGRID supports both methods but dynamic deployment is intended preferably for simple programs. It allows a quick employment of additional machines. The disadvantage is the organizational effort and the time required for setting up the applications at the target system. Due to the heterogeneous nature of the Grid, dynamic deployment is also error prone. AUGUSTUS is deployed using pre-installation because the software package also includes some species-specific parameter files and different runtime environments necessitate minor adaptations at each site. Additionally, there are security concerns of the resource providers and community that only authorized software releases are used in MediGRID.

After an appropriate machine or cluster has been chosen, the input data has to be transferred to the execution host. For data transfers, the Globus Toolkit provides GridFTP which works similar to the normal FTP and uses GSI for data encryption and credential-based access to the machines. An additional functionality of GridFTP is the capability to perform third party transfers. These are file transfers between two machines initiated by a third machine. GridFTP is a non-WS component and already existed in GT2. Its web service counterpart is called Reliable File Transfer (RFT) [44]. RFT requires a database such as PostgreSQL [45] in which it stores the status of the transmissions for fault recovery if the service is interrupted.

Once all preparations are done, the job can be submitted. The execution management service of GT4 is called WS-GRAM (Web Service Grid Resource Allocation and Management) [46]. WS-GRAM offers a variety of options. A standard command line to execute AUGUSTUS is:

```
globusrun-ws -submit -Ft PBS \
-F https://medigrid-srv.gwdg.de:8443/wsrf/services/\
ManagedJobFactoryService -so outputfile -c /opt/medigrid/\
augustus/bin/augustus --species=human example.fa
```

This is equivalent to the example of Section 2.1 and executes the program on the cluster of site GWDG. To this end, the job is submitted to the head node of the cluster called “medigrid-srv” which has the GT4 middleware installed. The working directory is the home directory of the Grid user, thus the input and output files are located there. Console output is written into the specified file (using the “-so” parameter). Several job managers (specified by the factory type parameter “-FT”) enable job submission to PBS, LSF or LoadLeveler resource management systems [47].

For more detailed job specifications, GT4 offers the XML-based Job Description language [48]. Its most important capability is to include file transfers in the job description. The transfers before and after job execution are called stage-in and stage-out. WS-GRAM relies on RFT to do these transfers. Further specifications allow setting the working directory, redirection of the console output and subsequent deletion of files. The job description in Fig. 2.2 executes AUGUSTUS with the same arguments as before, but also includes the necessary file transfers.

```
<job>
  <executable>/opt/medigrid/augustus/bin/augustus</executable>
  <directory>${GLOBUS_USER_HOME}</directory>
  <argument>--species=human</argument>
  <argument>example.fa</argument>
  <stdout>${GLOBUS_USER_HOME}/outputfile</stdout>
  <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
  <fileStageIn>
    <transfer>
      <sourceUrl>gsiftp://submissionhost/tmp/example.fa</sourceUrl>
      <destinationUrl>file:///${GLOBUS_USER_HOME}/example.fa</destinationUrl>
    </transfer>
  </fileStageIn>
  <fileStageOut>
    <transfer>
      <sourceUrl>file:///${GLOBUS_USER_HOME}/outputfile</sourceUrl>
      <destinationUrl>gsiftp://submissionhost/tmp/outputfile</destinationUrl>
    </transfer>
  </fileStageOut>
  <fileCleanUp>
    <deletion>
      <file>file:///${GLOBUS_USER_HOME}/example.fa</file>
      <file>file:///${GLOBUS_USER_HOME}/outputfile</file>
    </deletion>
  </fileCleanUp>
</job>
```

Figure 2.2: Example job description document.

It should be noted that all file locations are specified from the perspective of the execution host, not the submission host. The job is submitted with the following command line:

```
globusrun-ws -submit -F https://medigrid-srv.gwdg.de:8443/wsrf/services/\
ManagedJobFactoryService -f job-description.xml
```

As we can see, the Grid solves many of the problems that are encountered with cluster computing. A Grid user only needs one certificate to access all resources of the VO in the same way. WS-GRAM is a universal execution management service for the diverse local resource management systems on the sites of the VO. It makes data transfers less laborious by integrating them into the job description.

On the other hand, resource discovery and selection still have to be done manually with the Globus services. These are difficult tasks if there are many resource providers, and if a load balancing between the sites is desired. Furthermore, the above example only covers a single execution of AUGUSTUS. In case of splitting data for parallel processing, multiple program instances have to be run. Implicit data transfers are not performed automatically, and job monitoring remains a tedious task, as there is no user-friendly interface to check the status of submitted jobs. The significance of all these issues increases with the complexity of the job scenario. For example, if a computation consists of a workflow of parallel branches with subsequent program executions and intermediate data transfers, GT services are not sufficient anymore and workflow management capability is needed for high-level application steering. This automation can be provided by a workflow management system.

2.3 Running AUGUSTUS with the Workflow System

In MediGRID, the processing of complex application scenarios consisting of several program executions is handled by the Generic Workflow Execution Service (GWES) [17]. This workflow management system is used in many European Grid projects, e.g. MediGRID, PneumoGRID, Instant-Grid, BauVOGrid, K-Wf Grid and CoreGRID [22, 49, 50, 51, 52, 53].

The workflow management infrastructure consists of several components depicted in Fig. 2.3. The central service is the GWES which processes the workflow documents. GWES includes a ResourceMatcher that searches for available versions of the programs that are specified in the workflow on the machines in the Grid. Furthermore, GWES includes a Grid scheduler [54] that assigns job executions to the machines. For resource matching and scheduling, GWES communicates with an eXist resource database that contains information about the software versions available in the Grid. Additionally, the resource database contains machine descriptions with information about installed software and the current utilization of each machine. Both types of information are specified by means of the XML-based D-Grid Resource Description Language (D-GRDL) [55]. The machine descriptions are provided by ResourceUpdaters which are running as resource monitoring

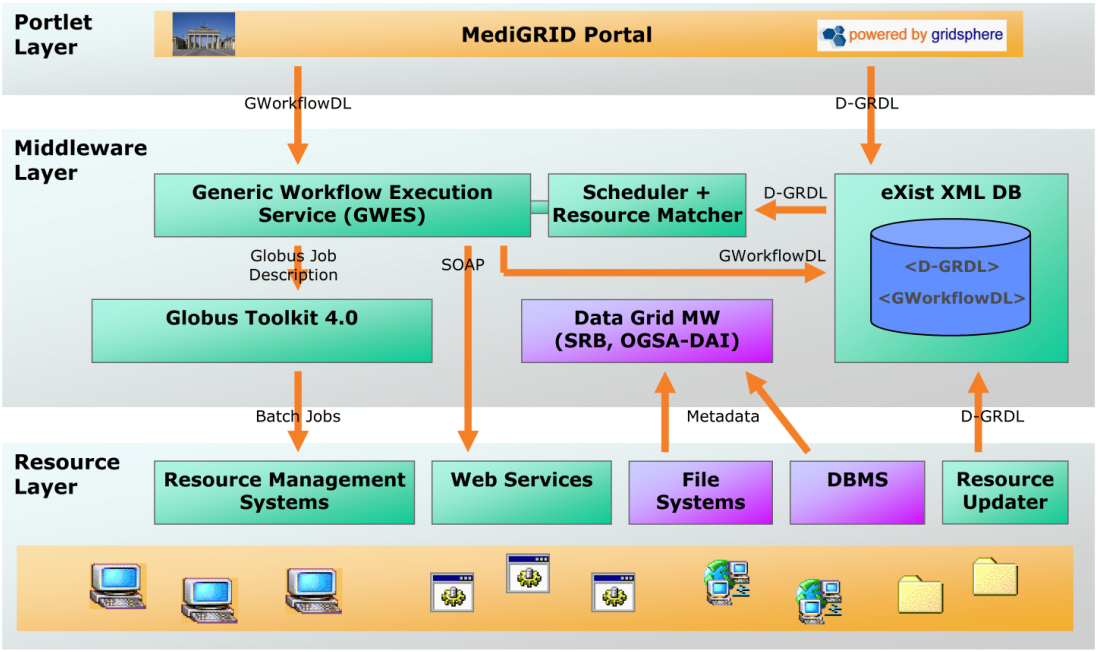

Figure 2.3: Architecture of the MediGRID middleware.

daemons on all machines. Finally, GWES also uses the eXist database to store workflow documents of active and completed workflows. This allows for checkpointing workflows, fault management and result provenance tracking

In contrast to most Grid workflow engines, the workflow model of GWES is based on Petri nets [56] instead of simple directed acyclic graphs (DAG). Petri nets are especially suited to describe distributed and concurrent processes. They consist of places (symbolized by circles) and transitions (symbolized by squares). Places and transitions are connected by directed arcs (directed edges) and always alternate with each other. The direction of the arc defines if a place is an input or output place of the transition. In the workflow, transitions stand for conditional activities while places represent input and output data. Places contain so-called tokens (symbolized by dots) as soon as the data are available. The distribution of tokens over the places is called a marking. The marking describes the state of the Petri net. Transitions can only be activated (also called "firing") if all input places of the transition are occupied by a token, and if all output places of the transition can receive a token. In this state the transition is called enabled. When an enabled transition is firing, one token from every input place of the transition is consumed, and one token is produced on every output place of the

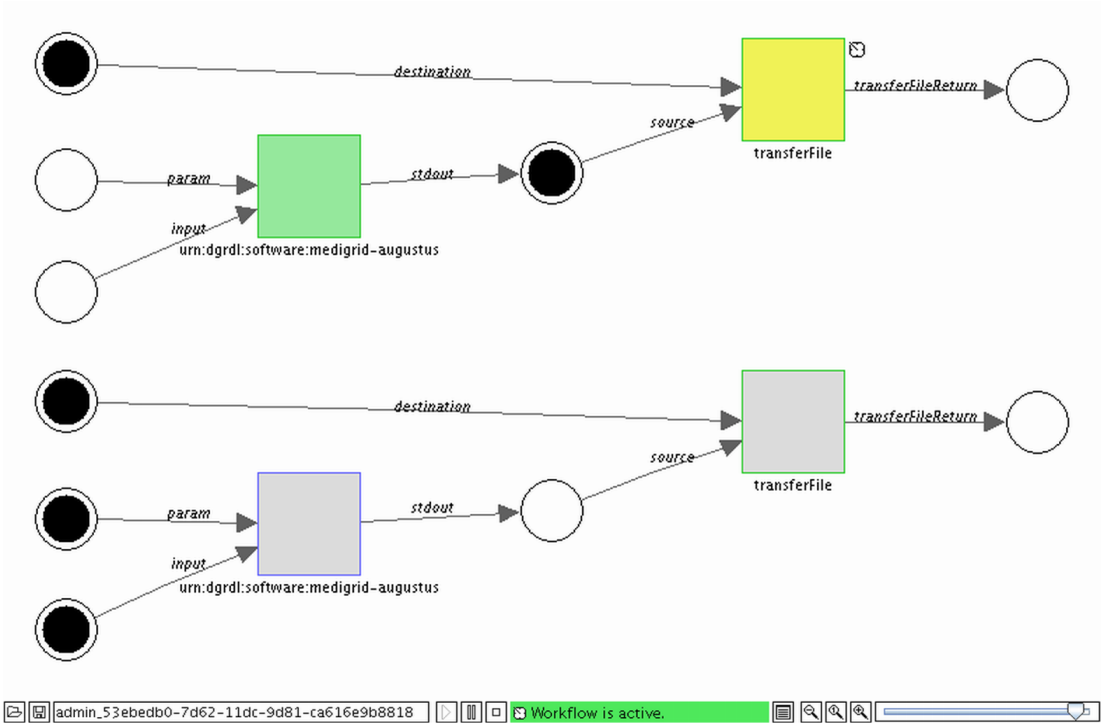

Figure 2.4: AUGUSTUS Petri net graph displayed in the GWES user interface.

transition. In this way, Petri nets allow to model state and actions of workflows, and control and data flow between application parts. The Petri nets of MediGRID workflows are specified with the XML-based Grid Workflow Description Language (GWorkflowDL) [57].

The first step to run AUGUSTUS with the GWES is to set up a workflow for the application. It is recommended to draw a Petri net graph at the beginning. The graph of a workflow with two parallel AUGUSTUS invocations is displayed in Fig. 2.4.

The second step is to convert the Petri net graph manually into the XML-based representation with GWorkflowDL. The GWorkflowDL description is necessary for interpretation by the workflow engine. The description consists of a set of XML tags which specify all the data places and transitions. The directed edges between places and transitions are specified within the transition clauses by the inputPlace and outputPlace tags. They connect the transition with the specified places. Additionally, every edge is named with a string called edgeExpression. A shortened description of the AUGUSTUS workflow is given in Fig. 2.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<workflow xmlns="http://www.gridworkflow.org/gworkflowdl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xsi:schemaLocation="http://www.gridworkflow.org/gworkflowdl
  http://www.gridworkflow.org/kwfgrid/src/xsd/gworkflowdl_1_0.xsd">
  <description>AUGUSTUS MediGRID Workflow</description>
  <property name="resource.repository.collection">/db/dgrdl</property>
  <place ID="fastainput0">
    <token><data>
      <file xsi:type="xsd:string">gsiftp://139.18.18.60//tmp/augustus/HS04636</file>
    </data></token>
  </place>
  <place ID="parameter0">
    <token><data><param xsi:type="xsd:string">--species=human</param></data></token>
  </place>
  <place ID="outputdestination0">
    <token><data>
      <param xsi:type="xsd:string">gsiftp://139.18.18.60//tmp/HS04636.gff</param>
    </data></token>
  </place>
  <place ID="augustusoutput0"/>
  <place ID="result0"/>
  <transition ID="augustus0">
    <description>Augustus worker 0</description>
    <inputPlace placeID="parameter0" edgeExpression="param"/>
    <inputPlace placeID="fastainput0" edgeExpression="input"/>
    <outputPlace placeID="augustusoutput0" edgeExpression="stdout"/>
    <operation>
      <pe:programClassExecution xmlns:pe="http://www.gridworkflow.org/gworkflowdl/programclassexecution"
        softwareClass="urn:dgrdl:software:medigrid-augustus"/>
    </operation>
  </transition>
  <transition ID="transfer0">
    <description>file transfer 0</description>
    <inputPlace placeID="augustusoutput0" edgeExpression="source"/>
    <inputPlace placeID="outputdestination0" edgeExpression="destination"/>
    <outputPlace placeID="result0" edgeExpression="transferFileReturn"/>
    <operation>
      <ws:WSClassOperation xmlns:ws="http://www.gridworkflow.org/gworkflowdl/wsclassoperation">
        <ws:WSOperation
          wsdl="http://portal.medigrid.izbi.uni-leipzig.de:9081//gwes/services/FileTransfer?wsdl"
          operationName="transferFile" selected="true"/>
      </ws:WSClassOperation>
    </operation>
  </transition>
</workflow>
```

Figure 2.5: GWorkflowDL workflow description for AUGUSTUS with a single input sequence.

The operation tag within the transition clause denotes what activity will be performed by the transition. As the workflow description shows, GWES supports two types of operations, program executions and web service invocations. The first transition executes the AUGUSTUS command line program, while the second transition performs an explicit file transfer to a specified destination. At present, the invoked web services are explicitly specified in the workflow description by their Uniform Resource Identifier.

In the following we focus on program executions. In order to set up the workflow, the user has to specify an abstract program class only. When the workflow is passed to the GWES either by the user or automatically (see Section 2.4), GWES initializes the workflow and does a resource matching. This means, it searches for available instances of the program classes that are used in the workflow. With program instance we denote a pre-deployed program on a machine. For this, MediGRID employs a resource XML-database that contains information about the existing program instances and the software installed on each machine. Both types of descriptions are specified with the XML-based D-GRDL.

During resource matching, GWES first queries the resource database and retrieves a list of all program instances matching the abstract program class. This requires a software resource description which specifies what software class a program instance belongs to. It also contains the path to the executable file. Thus, the next step in gridifying is to provide the software resource descriptions of all different program instances. For servers with an identical software version installed at the same path the same D-GRDL description applies. An example software resource description for AUGUSTUS is given in Fig. 2.6.

In the second step of resource matching, GWES retrieves a list of all machines where one of the instances is installed. This requires the hardware resource description which lists all program instances provided by the machine. Furthermore, the machine description contains static machine information and utilization data. The hardware resource descriptions of the machines in the Grid are created automatically by the ResourceUpdater daemon installed at each machine. The ResourceUpdater stores the description in the resource database and updates the utilization data periodically. An example hardware resource description for the cluster of site GWDG with the head node “medigrid-srv” is given in Fig. 2.7.

Based on the current utilization of the machines, GWES automatically selects

```
<?xml version="1.0" encoding="UTF-8"?>
<resource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.gridworkflow.org/kwfgrid/\
  src/xsd/resource-d-grdl.xsd"
  uri="software:medigrid-augustus-2-0">
  <ofClass uri="urn:dgrdl:software:medigrid-augustus"/>
  <name>Augustus</name>
  <description>AUGUSTUS gene prediction program</description>
  <simpleProperty ident="executable" type="string">
    /opt/medigrid/augustus/augustus.sh
  </simpleProperty>
</resource>
```

Figure 2.6: Software resource description for AUGUSTUS.

hardware resources for the execution of the jobs in the workflow. This step is the scheduling in MediGRID which we explain in more detail in Section 5.1. After the resource selection is completed by GWES, the programClassExecution tag contains a sub-entry such as:

```
<pe:programExecution
hardware="hardware:medigrid-srv.gwdg.de/PBS"
software="software:medigrid-augustus-2-0"
quality="0.75" selected="true"/>
```

Now the Petri net description can be processed by GWES. GWES provides automatic file stage-in, i.e. it organizes all necessary data transfers so that input data are available at each machine on which a job is scheduled and ready to execute. For these file transfers, GWES uses the RFT service from the underlying GT4 middleware.

To be executed by GWES, command line programs have to comply with a common parameter syntax. Therefore, executables may have to be encapsulated with wrapper shell scripts, because GWES passes arguments such as input and output files and program parameters in always the same way. A listing for an example wrapper script is given in Fig. 2.8. The parameter names of the wrapper script correspond with the edgeExpressions in the workflow. Command line output can be redirected using stdout and stderr. The job submission is done via WS-GRAM, the job manager of GT4. For this purpose, GWES automatically creates

```
<?xml version="1.0" encoding="UTF-8"?>
<resource xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.gridworkflow.org/kwfgrid/\
  src/xsd/resource-d-grdl.xsd"
  uri="hardware:medigrid-srv.gwdg.de/PBS">
  <ofClass uri="urn:dgrdl:hardware"/>
  <name>medigrid-srv.gwdg.de/PBS</name>
  <description>Hardware resource medigrid-srv.gwdg.de with local\
    resource manager PBS and Globus Toolkit
  </description>
  <provides>
    <!-- tests and benchmarks -->
    <resourceRef uri="software:cat-medigrid"/>
    <!-- utils -->
    <resourceRef uri="software:medigrid-utils-chmod-all-read-0-1"/>
    <resourceRef uri="software:medigrid-utils-show-environment-0-1"/>
    <!-- augustus -->
    <resourceRef uri="software:medigrid-augustus-2-0"/>
  </provides>
  <simpleProperty ident="Info.LRMSType" type="string">PBS</simpleProperty>
  <simpleProperty ident="Info.GRAMVersion" type="string">4.0.3
  </simpleProperty>
  <simpleProperty ident="Info.ContactString" type="uri">
    https://medigrid-srv.gwdg.de:8443/wsrf/services/ManagedJobFactoryService
  </simpleProperty>
  <simpleProperty ident="State.RunningJobs" type="int">5</simpleProperty>
  <simpleProperty ident="State.WaitingJobs" type="int">6</simpleProperty>
  <simpleProperty ident="State.TotalJobs" type="int">11</simpleProperty>
  <simpleProperty ident="gwes.gram.home.directory" type="string">
    /opt/medigrid/tmp
  </simpleProperty>
  <simpleProperty ident="score">1911</simpleProperty>
</resource>
```

Figure 2.7: Hardware resource description for medigrid-srv.

```
#!/bin/bash
#
# AUGUSTUS wrapper script
#
# program location
MEDIGRID_HOME = /opt/medigrid
AUGUSTUS=$MEDIGRID_HOME/augustus
AUGUSTUS_BIN=$AUGUSTUS/bin/augustus

# get arguments
PARAM=${2}  # get 2nd argument (string)
INPUT=$4    # get 4th argument

# check if number of arguments equals 4
if [ $# -ne 4 ]; then
  echo "### Usage: $0 -param "parameters" -input inputfile"
  exit 1
fi

# check if AUGUSTUS binary exists
if [ ! -r "$AUGUSTUS_BIN" ]; then
  echo "### Error: executable 'augustus' not found"
  exit 1
fi

export AUGUSTUS_CONFIG_PATH=$AUGUSTUS/config
exec $AUGUSTUS_BIN $PARAM $INPUT
echo "### AUGUSTUS done ###"
```

Figure 2.8: AUGUSTUS wrapper script for execution on Grid machines.

the WS-GRAM job descriptions to submit Globus jobs that in turn execute the wrapper scripts. If a job is not finished successfully, the fault management of the workflow system automatically selects another suitable resource and submits the job again.

Program execution operations are very suitable to gridify so-called legacy applications for GWES. Because of the wrapper scripts, the gridification is independent of the functional application development. It does not require modifications of the application source code and only has to be done once. This is important for AUGUSTUS, as the application is under continuous development.

Alternatively it would also have been possible to gridify AUGUSTUS as a web service. However, this requires either a laborious reimplementation of the application as a web service or a web service wrapper that might have to be continuously adapted to application development. Additionally, web services necessitate web service containers which are usually not available on cluster nodes.

The GWES software package also includes a graphical interface called Grid Workflow User Interface (GWUI) which allows the user to monitor and steer the processing of the workflow (see Fig. 2.4). GWUI is implemented as a Java applet and accessed via a web browser. It displays how the Petri net tokens move through the workflow graph as the transitions are executed. The menu bar at the bottom shows the current workflow state and contains buttons to suspend, resume or stop the workflow. Furthermore, the current state can be stored in GWorkflowDL format. By clicking into the graph the properties of tokens, places and transitions can be inspected and manipulated. For transitions e.g. it is possible to select one of the available resources manually.

Most of the problems that exist with the basic Globus middleware are solved with the use of the workflow system. Once an application is gridified for GWES, the complete execution process is automated. This includes resource discovery and selection, implicit data transfers between job executions as well as the job submission. GWES provides a much higher quality of service than GT4 and delivers the scalability and performance for complex application workflows comprising of parallel computational branches. Additionally, GWUI offers an improved usability and better overview of the status during workflow execution.

However, some effort remains as workflow descriptions have to be created for every application run. Furthermore, users are required to have Grid knowledge and direct access to a Grid machine. It is therefore advisable to provide the control of the application within a graphical user interface (GUI) that can be accessed with a web browser. Such a GUI can be realized with a Grid-enabled web portal.

2.4 Creating a Graphical User Interface

MediGRID uses GridSphere [58] which is a framework to establish Grid portals. GridSphere provides a central access to Grid services via a single world-wide

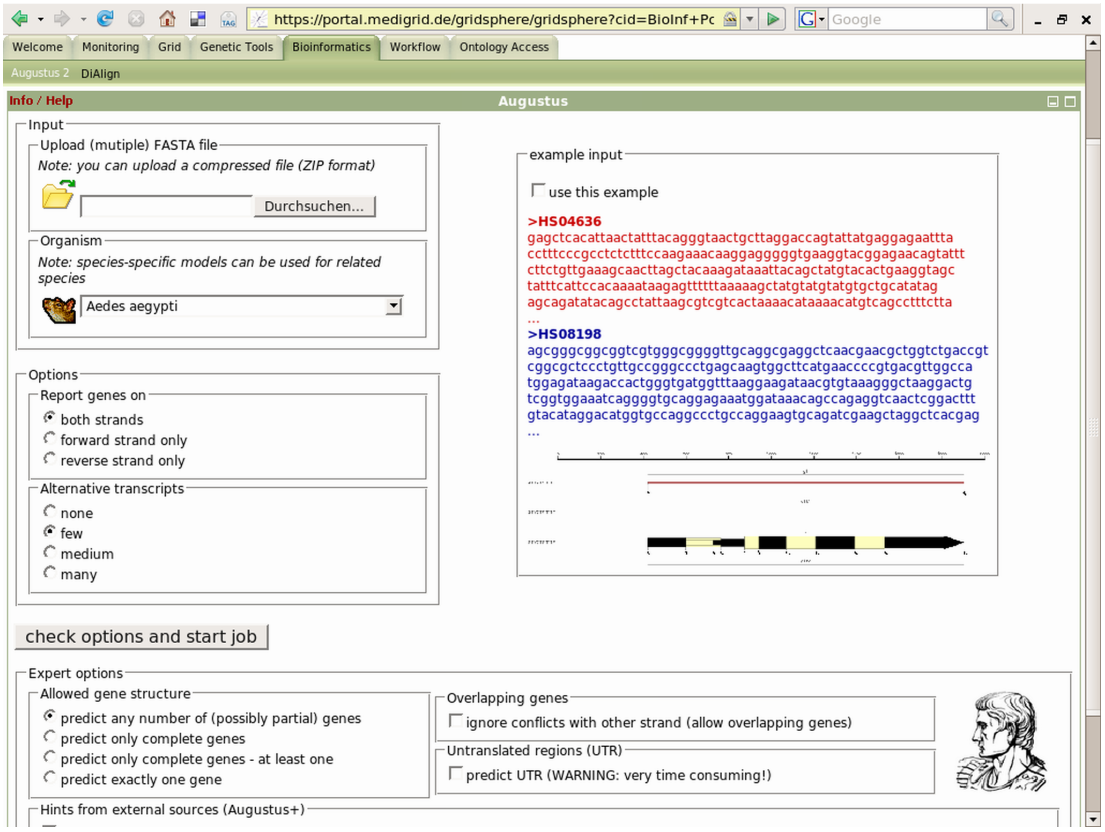

Figure 2.9: Screenshot of the AUGUSTUS application portlet within the MediGRID portal at https://portal.medigrid.de.

accessible user interface. Inside a Grid portal, small applications can run which are called portlets [59]. In MediGRID, the portlets provide access to services from Globus and MediGRID developers. Each application has its own portlet as an easy-to-use graphical interface.

The development of a GridSphere portlet starts with the creation of a GridSphere project which sets up a project directory and some basic portlet configuration files in XML format. After editing the configuration files, the portlet user interface and logic are developed by means of Java Server Pages (JSP) [60] and a portlet Java class, respectively. The AUGUSTUS portlet is based on GridSphere's action provider model. The action provider model and the associated action portlet interface provide the possibility to separate the graphical layout from the portlet logic. The GUI consists of several JSPs which contain the layout description in HTML, and action components from GridSphere's user interface library. These

components, e.g. links, buttons and form fields, can be accessed within the Java class. An application usage scenario is implemented as a series of requests and responses.

The AUGUSTUS portlet consists of three main stages: job configuration, workflow execution, and result presentation. Within the job configuration stage (see Fig. 2.9), the user can upload a multiple FASTA sequence file and configure some basic AUGUSTUS prediction parameters. In order to incorporate hints from a BLAST search, the user only has to activate an additional checkbox.

Usually users do not access the portlet from a computer within the Grid, thus the input file has to be transferred from the user's computer to a Grid machine. This is necessary because the file must be accessible via RFT to enable further transfers with GT4 middleware. At present, the upload is done via HTTPS and the portal server is used for temporary storage. Another option would be to upload the input file to a Grid storage element with a suitable upload client. When the user starts AUGUSTUS from the portlet, the parameters are retrieved from the user interface components and stored in a parameter string.

The splitting of the input file according to the number of contained single sequences is done on the portal server. On one hand, splitting is very fast in comparison to gene prediction. On the other hand, splitting on an arbitrary Grid machine would require splitting software on this machine and additional data transfers within the Grid. Furthermore, the input file would have to be parsed twice because the workflow has to be specified entirely within the portlet before invocation. The single sequences are stored as files in a temporary directory on the portal server. The file locations are specified within the data tokens on the input places of the Grid workflow.

In the workflow execution stage, the GWorkflowDL workflow as described in Section 2.3 is automatically created using the StringTemplate Java library [61]. For the creation of the workflow description, a template for a single AUGUSTUS invocation is duplicated according to the number of sequences, and the variables within the template are replaced by the actual filenames. When hints from a BLAST search are to be considered, a different workflow template is used which contains an extra transition for the hint search for each input sequence. The resulting workflow description has to be concatenated to the corresponding header before execution.

After the Grid workflow has been created, it can be initiated using the GWES Java API. Then, GWES processes it according to the specified steps and returns either an execution success or an error. The process of the workflow execution can be monitored by means of the GWUI Java applet. With this applet, the graphical interface of the workflow engine can conveniently be integrated into the portlet to provide the user with progress and status information.

The last transition in the AUGUSTUS workflow transfers the results back to the portal server where they are automatically postprocessed after successful workflow completion. The postprocessing consists of a transformation of the prediction result of every sequence into a graphical representation. If the required software is installed on the Grid nodes, this process can also be integrated into the workflow. The prediction results are displayed in the result stage of the portlet and can also be downloaded in a compressed format.

Portals such as in MediGRID provide the possibility to connect related applications so that the results can be further analyzed. Connections to the related ontology access portlet in the MediGRID portal are integrated into the AUGUSTUS portlet via so-called actionlinks from GridSphere's user interface library. This allows the user to access information about specific sequence regions. Furthermore, predicted genes can automatically be examined for similar sequences in databases. This is an advantage over the stand-alone application because without the possibilities of the portlet the user would have to extract the coding sequences from the prediction results and perform the search manually.

2.5 First Results and Lessons Learned

The gridified version of AUGUSTUS is available for end users in the MediGRID portal. At the time of our first evaluation, the application could utilize MediGRID resources comprising of eight systems at five different sites with more than 1000 processors. Since then, additional resources were gradually added by deploying the application at further sites and providing GWES with the resource descriptions and monitoring data. Due to the roughly linear scaling of execution time with the number of CPUs this establishes processing capabilities to decrease the execution time considerably. To evaluate the speed-up compared to the non-gridified version, we compared the overall execution time of AUGUSTUS

in MediGRID and a stand-alone version on a single Pentium 4 PC with 2.8 GHz CPU frequency and 1 GB RAM. For this purpose, we used the genome of the green alga Chlamydomonas reinhardtii which contains 88 sequences and roughly 114 million base pairs in total. While the computation on the Desktop PC took 1842 minutes, the grid jobs required 64 to 114 minutes depending on the total grid load at the different start times. This corresponds to a speed-up of 16 to 29, which is an order of magnitude faster than the stand-alone version. However, the theoretical speed-up of approximately 1000 was not achieved for several reasons. First, usually only a small part of the resources are instantly available for computation at the particular job submission time because most resources are well utilized. Second, the workflow management always incorporates some time overhead, especially when program executions fail. In our workflow examples several program executions failed because of errors on particular resources. Nevertheless, the workflows completed successfully due to the fault management mechanism implemented in GWES.

An important lesson learned from the experiments was that faults in job executions in D-Grid were much more frequent than expected. Therefore, fault management is a very important issue in gridification. Some of the applications that extensively use D-Grid computing resources rely only on basic GT4 technology to reduce the possible points of failure. However, in these cases developers and end users are usually identical. For applications that are intended to be used by e.g. biomedical researchers, a user-friendly approach with a portal such as in MediGRID is necessary. As we motivated, such an approach requires the employment of several middleware layers. Because debugging in this environment can be very tedious, it is preferable to successively develop functional components in a bottom-up manner.

Some D-Grid applications are already capable to continuously occupy all D-Grid resources. Therefore, an improved scheduling is required on the sites as well as Grid-wide. At site level, mechanisms like fairshare algorithms should be used to achieve a balanced usage among the communities. At the Grid level, the use of a meta-scheduler like in GWES allows for load-balancing between several sites.

For the design of workflows it is advisable not to aim at a very fine-grained structure, because delays introduced by waiting times can considerably extend the overall completion time of jobs. User interfaces as known from classical applica-

tions have to be adapted carefully to Grid applications, because the possibility to process larger amounts of data can necessitate special mechanisms for data input and result presentation.

Although some regulations on resource configuration exist in D-Grid, a considerable extent of flexibility regarding the software stack of sites remains. Therefore, developers need support from the computing centers to solve site-specific issues. As with traditional cluster computing, computing centers have to assist in porting the application to their specific runtime environments, and they must install application and middleware components. Furthermore, they can provide Grid coaching for developers and users on the middleware layers of the Grid.

2.6 Summary Discussion

In this part of the thesis we have described the adaptation of applications to Grid computing. We term this process gridification. It breaks down into steps that depend on the middleware that is employed in the Grid. In MediGRID this are Globus Toolkit 4, the GWES workflow system and the portal framework.

The process starts with parallelization of the application that allows for distribution of the computation. Often data parallelism can be exploited which simplifies distribution. For the transition to the Grid the developer has to obtain a user certificate that enables authentication. Afterwards, the developer must become a member of the VO to get the authorization to access the Grid's resources. After deployment of the application on the machines, the functionality should be verified by execution using the Globus services. This involves the creation of a formal job description for the application.

To run the application with GWES, a schematic application workflow must be set up followed by the formal workflow specification in the workflow description language. Afterwards, the software resource description has to be created to enable resource matching. The adaptation of the application to the generic invocation syntax of the workflow system is achieved with wrapper scripts for the command line programs. In the end, correct workflow execution can be verified via the GUI provided by GWES.

The development of the application-specific portlet for the MediGRID portal

starts with creation of a GridSphere project and editing the portlet configuration. Then, the portlet user interface is created with several JSPs. The portlet logic is implemented in Java. Its main tasks are the creation of the workflow description from a template and to initiate the workflow processing using the GWES API. Finally, the portlet is deployed on the portal server and can be accessed for testing.

At the end of the gridification process the application is provided as a service which can be accessed simply with a web browser. The user interface corresponds to the scientific domain of the end users, i.e. it uses the terms of the respective field. End users only have to login at the Grid portal to utilize the Grid's resources. They do not need detailed Grid knowledge. This way, gridification provides accessibility and usability.

We have shown that Grid technology helps to reduce the execution time of the gene prediction program AUGUSTUS. This improvement in performance allows to tackle larger problem sizes which cannot be accomplished with limited local resources. Additionally, the extended functionality of the portlet facilitates the further analysis of the results a lot in comparison to the stand-alone application.

Part II

Grid Workflow Scheduling

Chapter 3

State of the Art

The scheduling of scientific workflow applications on the Grid is a challenging task that has been subject of research for many years. Many research groups have contributed to the scientific progress in this domain and the number of approaches is as numerous as the projects dealing with the topic. In this Chapter we will give an overview of the current state of the art. The experience we gained during the gridification of AUGUSTUS (cf. Section 2.5) as well as detailed measurements in D-Grid (cf. Section 4.2) showed that efficient Grid workflow scheduling requires information about the input-queue waiting-time of Grid jobs. Therefore, we start with a look at related works on queue waiting time prediction before we present related works on Grid workflow scheduling.

3.1 Queue Waiting Time Prediction

The prevalent compute resources in D-Grid are parallel computers that are built up as clusters. These clusters are located at the computing centers that take part in D-Grid. Although some clusters were set up to serve mainly specific D-Grid communities most of them are generally available for all VOs in D-Grid. Clusters consist of many computing nodes with multiple processors all of which are controlled by a local resource management system (LRMS). The typical operation principle of clusters is the so-called space sharing. In space sharing a dedicated set of processors is allocated to each job during the time of its execution. If all processors are allocated to running jobs, new jobs that arrive at the cluster must

wait in input queues. The LRMS manages the jobs in the cluster during their life cycle. Particularly, it calculates priorities for all waiting jobs and schedules them onto the processors in the order of their priority. Sometimes the LRMS is also denoted as batch system to emphasize that jobs are processed non-interactively in batch mode.

Meta-scheduling in Grids distributes jobs across matching compute resources based on the resources' current availability. For cluster resources the key parameter is the queue waiting time of jobs. This is the duration between the instant of time when the job is created in the queue and the instant of time when the processing starts. The majority of previous works on queue waiting time prediction is based on statistical analysis of past workloads for job lengths, followed by a simulation of the local scheduling to determine the start times of the jobs in the system.

Downey [62, 63] presents statistical techniques for predicting the queue time of jobs on space-sharing parallel machines with first-come first-served (FCFS) scheduling. The focus of his work is restricted to the special case of moldable jobs. Such jobs can use a variable number of processors which is configured before the job starts and remains constant during execution [64]. In his approach the decision is taken when the job arrives at the head of the queue. The aim is to choose that number of processors that minimizes the sum of (further) queue time and processing time. For this, the processing times on a range of cluster partitions and the waiting times until the partitions become available have to be predicted.

The queue time prediction uses a log-uniform distribution to model remaining lifetimes of jobs in all partitions based on their current age. The aggregation of the job-by-job predictions then yields the respective start time of the job at the head of the queue. Downey's work does not address the prediction of the entire queue waiting time of a newly submitted job. This, and the premise of FCFS scheduling limit its wider applicability.

Smith, Taylor and Foster [65] describe a prediction methodology for the complete time jobs have to wait until their execution starts. Their technique uses predictions of job execution times along with a simulation of the scheduling algorithms to determine when jobs will begin to execute. The prediction of execution times is based on a categorization of jobs with templates that describe their attributes,

and deriving job execution times from execution times of similar jobs in the past. For this purpose historical information is employed comprising workload traces of four parallel computers. From the execution times, the authors calculate mean waiting times for every job in the system by simulating the actions of the scheduler. The supported scheduling algorithms are FCFS, least work first and backfill [66].

Additionally, Smith [67] also considers prediction of job waiting times from historical information on the scheduler state. This method employs the categorization technique he uses for predicting job execution times directly on prediction of waiting times. It uses templates that describe e.g. the time the prediction is made, the jobs currently running on the machine, and the number of jobs that are waiting in the queue ahead and behind of the job being predicted. The prediction is derived from past job waiting times during similar scheduler states. However, in comparison this method performed worse than the first prediction technique.

Li et al. [68] describe another approach based on statistical analysis of historical job traces and simulation of site schedulers. They use historical job information to predict execution times of currently running and queued jobs at the site. As an extension to the work of Smith et al. they further investigate the selection of templates that yield the best characterization of similarity between jobs. They propose a method that does not only select a single template for prediction but also considers predictions based on average estimates over a set of templates. For the simulation of the scheduling algorithms they employ the Maui scheduler [66] which supports a number of algorithms and has a simulation mode. However, this simulation lacks performance because it is not event-driven which is due to the complexity that results from dynamic scheduling policies like fairshare. To improve performance, the Maui simulation was made event-driven with the downside that it no longer supports dynamic policies. Input to the simulation are the predicted job execution times and the configuration file with the site's scheduling policy. The policy can define specific priorities for groups (VOs) and/or users, therefore the simulation yields predicted job start times dependent on the group and/or user.

The simulation based approaches by Downey, Smith et al. and Li et al. can provide accurate waiting time predictions for certain scheduling algorithms and accurate job execution time predictions. However, there are a number of limi-

tations. All approaches require detailed knowledge of the scheduling algorithm. Downey and Smith et al. additionally assume that the scheduler employs a rather simple algorithm, and that all users, groups or queues have equal priorities. While Li et al. do not impose this restriction, they still assume that the targeted sites do not employ dynamic policies e.g. with usage based prioritization. Furthermore, all works assume that the scheduling algorithm and resource pool are static and do not change over time. A general limitation of simulation based approaches is that waiting time prediction will not be very accurate if the scheduling algorithm is such that waiting time depends on jobs that have not yet been submitted to the system. Finally, for the prediction of job execution times all methods depend on the availability of historical workload traces as well as information about all jobs that are currently running and waiting in the system. Unfortunately, most of these conditions are not met in the D-Grid environment (cf. Section 4.2).

Brevik, Nurmi and Wolski [69, 70, 71] consider these limitations and present an alternative approach to queue waiting time prediction that is not based on simulation. Instead they propose a statistical method that infers job waiting times only from historical job waiting time data. This has the advantage that neither prediction of job execution times nor assumptions about the scheduling algorithm are necessary. The second difference from the previous works is that their approach does not calculate point-valued predictions of the mean waiting times. Instead it yields upper (and lower) bounds on job waiting times in input queues together with a measure of confidence. Their approach which they term the Binomial Method Batch Predictor (BMBP) addresses the problem of determining "the maximum amount of time a job is 'likely' to wait in a batch queue before it is executed" [69] with likely meaning that e.g. 95% of the predictions will be at least as great as the actual waiting times. BMBP regards job waiting time as a random variable, so that this amounts to finding an upper bound on the 95th percentile, or 0.95 quantile, of this variable's distribution. However, since the input of the prediction is a historical trace of waiting times that jobs have experienced in a queue, the true distribution of waiting time is unknown. To compensate, BMBP determines the upper bound according to a defined confidence level which expresses the certainty that the upper bound is really at least as great as the quantile. The confidence level affects the size of the historical trace that is used for prediction. A higher confidence level demands a higher number of historical values, which brings the downside that prediction reacts slower on change-points

in waiting time. Both, quantile and confidence level can be adjusted by the user. BMBP typically uses the 0.95 quantile and a confidence level of 95%.

The BMBP approach has been developed further into QBETS (Queue Bounds Estimation from Time Series) [72]. In this work the authors position their approach as a forecasting methodology for time series albeit it does not consider correlation between data points to improve forecasting which is the basis of traditional time series analysis [73]. The QBETS system is used for queue waiting time prediction in the US TeraGrid [74]. For its predictions, QBETS acquires the latest trace data from the site scheduler. Therefore, it must be set up by site staff as such data access typically requires administrative privileges. Sonmez et al. [75] analyzed the performance of QBETS as well as of a simulation based point-valued predictor. In their evaluation QBETS performed better than another upper-bound method but had a lower average accuracy than the point-valued predictor with three out of four examined workload traces from the Grid Workloads Archive [76].

The probably best known performance monitoring and forecasting service for Grid environments is the Network Weather Service (NWS) by Wolski et al. [77, 78]. It does not feature queue waiting time prediction (QBETS originates from the same research group) but focuses on predicting network throughput and CPU availability. However, we reference it because of its relevance in the domain of statistical forecasting of time series data. The NWS infrastructure covers the three fundamental functionalities monitoring, forecasting and reporting. The monitoring is performed by so-called sensors that generate timestamp, performance measurement pairs. Measurements typically occur periodically with a trade-off between intrusiveness and accuracy. The NWS forecasting follows the idea that the best forecasting method amongst a number of available methods can be selected based on past accuracy. It is split into a first and second level that employ primary and secondary forecasters respectively. The primary forecasters generate predictions based on time series data. They comprise completely non-parametric methods such as last measurement, mean and median. Additionally, there are sliding window average, adaptive window median, and exponential smoothing methods that regard more recent data as more relevant to the prediction. The NWS operates all of the primary forecasters simultaneously. The secondary forecasters are used to dynamically select the primary forecaster with the lowest cumulative error

regarding mean absolute error and mean square error. The prediction of that primary forecaster then becomes the final forecast. For reporting NWS supports multiple interfaces to allow compatibility with a wide range of Grid scheduling and resource allocation systems.

Our approach to queue waiting time prediction which is presented in Chapter 4 is based on time series analysis. It uses an architecture similar to that of NWS but incorporates additional, novel prediction and selection methods.

3.2 Grid Workflow Scheduling

Grid workflow scheduling unites two fields of research that were originally investigated separately. The first field is that of meta-scheduling jobs across distributed Grid resources. The second field is the task [1] scheduling for workflows. Due to this, much of the related work focuses either on the domains of meta-scheduling or workflow scheduling. In the following, we will shortly address both domains and then give an overview of the Grid workflow scheduling problem and categorize the relevant workflow systems for Grids.

Grid scheduling is part of the Grid resource management and must be seen in this context because it has connections to many services in the Grid middleware, e.g. information services, execution management and data management. A comprehensive overview of Grid resource management is given by Nabrzyski, Schopf and Weglarz in [79]. Grid scheduling is concerned with mapping jobs to resources which are distributed over multiple administrative domains. The process of Grid scheduling can be divided into three phases: resource discovery, system selection, and job execution [80]. Resource discovery employs the information services to determine all resources for which an authorization exists. Their specification is then compared to the requirements of the job. This step is also denoted as resource matching. The outcome is a set of potential resources. In the system selection phase the scheduler obtains the latest resource performance data from the monitoring. Based on this information and its scheduling policy, the scheduler selects one or several resources to execute the job. This decision making can be considered the actual scheduling action. In the job execution phase the scheduler

[1] We use the term task in the context of workflows. Workflow tasks become jobs when they are executed on the Grid.

relies on the execution management services which may reserve resources in advance and perform the job submission. Transfer of input and output is performed by the data management services. The scheduler controls these actions and monitors the job execution. Upon successful completion the execution phase ends with cleanup operations otherwise the scheduler may induce a fault handling.

Dong and Akl [81] present an overview of Grid scheduling algorithms. They describe a taxonomy that differentiates at the top level between static scheduling which makes all scheduling decisions at one point in time, and dynamic scheduling which delays scheduling decisions to the latest possible time. The dynamic approaches again divide into centralized and distributed schedulers which can be either cooperative or non-cooperative. The final differentiation for all categories is between optimal and sub-optimal algorithms. The application of the taxonomy on existing schedulers shows that the vast majority of them falls into two groups. The first is that of static, sub-optimal algorithms, the second group is dynamic, centralized, sub-optimal algorithms. In both groups sub-optimal scheduling is typically based on heuristics.

Another characteristic that Dong and Akl examine is the objective functions of scheduling approaches. They separate these into application centric or resource centric optimization. Application centric algorithms reflect the users' perspective and typically try to minimize the execution time and/or the economic cost. As the Grid architecture becomes service oriented, some approaches also regard quality of service as a scheduling criterion. The resource centric algorithms reflect the perspective of resource providers. Such algorithms try to maximize resource utilization, throughput and/or economic profit.

A common Grid scheduler is the GridWay meta-scheduler [82] since it is part of the latest releases of Globus Toolkit 4. GridWay integrates with the Globus middleware and uses its information, execution management and data management services. It belongs to the group of dynamic, centralized, sub-optimal algorithms and supports job centric as well as resource centric scheduling policies [83]. However, GridWay's scheduling capabilities greatly depend on the quality of resource information. Therefore, it struggles with space-sharing clusters, because Globus' information service only provides insufficient information about such resources.

In addition to running independent jobs, GridWay also incorporates support for workflows that are based on directed acyclic graphs (DAG). For this purpose it

employs Condor DAGMan [84] which handles the inter-job dependencies. However, the scheduling is still performed on a per job basis.

The scheduling of all tasks in a workflow graph at the same time, also denoted as the task scheduling problem, is a traditional research discipline in parallel computing. The objective of workflow scheduling is to determine a mapping of tasks to processors that complies with the task dependencies and minimizes the overall completion time. This workflow execution time is also denoted as makespan.

The task scheduling problem is typically addressed with heuristic scheduling algorithms. Among these, especially list scheduling gained attention because it is practical in use and provides good performance results at low scheduling times [20].

In general, list scheduling algorithms construct an ordered list of tasks by assigning a priority to each task. When tasks are ready to execute, they are selected according to their priority (highest first) and assigned to the processor that minimizes a predefined cost function. The differences between the algorithms relate to the definition of the priorities and the employed cost functions. Most list scheduling heuristics are designed for systems with homogeneous processors. An algorithm for heterogeneous processors is the Heterogeneous Earliest-Finish-Time algorithm (HEFT) proposed by Topcuoglu, Hariri and Wu [20].

When applied to Grids environments however, list scheduling and other conventional workflow scheduling approaches become less efficient. This is because they originate from cluster systems with centralized control and internal networks that offer little latency and high throughput. Additionally, on clusters the resource pool can be assumed invariant. These characteristics do not apply to Grids, therefore Grid workflow scheduling developed as an independent field of research.

All Grid workflow schedulers that are oriented towards applicability in production environments are integrated with workflow management systems. An overview of such systems is given by Yu and Buyya [85]. They propose a taxonomy for Grid workflow management systems and categorize a number of approaches.

Another survey on Grid workflow approaches is presented by Wieczorek, Hoheisel and Prodan [86, 87]. It focuses stronger on the integrated schedulers and provides a comprehensive analysis of Grid workflow scheduling. In their survey the

authors introduce a set of five taxonomies which represent different "facets" of the workflow scheduling problem. We will adhere to these taxonomies to give an overview of Grid workflow systems.

The first taxonomy characterizes the underlying workflow models. It differentiates workflow systems according to their component model which can be task oriented or task and data transfer oriented. In the latter, tasks and data transfers are represented by nodes in the workflow graph. The workflow model itself can be based on DAGs or even simpler graphs, however some systems also employ more expressive models like Petri nets. Another aspect is the changeability of workflows which are denoted as "tunable" if nodes may be added or removed by the scheduler. Finally, Wieczorek et al. classify workflow models according to their data processing. In the pipelined model, workflows are repeatedly executed to process a stream of input data.

The second taxonomy regards the scheduling criteria. Many workflow systems employ schedulers that are able to optimize multiple criteria at the same time. Specific criteria comprise the execution time, economic cost, reliability and data quality. Additionally, Wieczorek et al. classify systems that support Grid-oriented criteria (e.g. resource usage, fairness, profit) or generic criteria. Finally, the costs related to criteria may be fixed or adaptive due to negotiations.

The third taxonomy addresses the scheduling process. One of its aspects is if multiple workflows can be scheduled at the same time. Another differentiation regards the dynamism, i.e. the time when the scheduling is performed. Just-in-time approaches schedule tasks when they are ready to execute, while full-ahead approaches schedule the whole workflow before the execution starts. This relates to dynamic and static scheduling described by Dong and Akl. The third option are hybrid approaches that combine just-in-time and full-ahead scheduling. Finally, Wieczorek et al. distinguish workflow systems with and without support of advance reservations.

The fourth taxonomy is concerned with the supported resources. Naturally, Grid workflow schedulers should be able to handle heterogeneous resources. A rare case is support of multiprogrammed resources. Such resources typically use multitasking to concurrently execute multiple workflows tasks.

The fifth taxonomy characterizes the workflow tasks. Workflow systems might offer support for moldable and/or malleable tasks. As described in Section 3.1,

moldable tasks can use a variable number of resources which is set before the execution starts. Malleable tasks use a variable number of resources during their execution. Additionally, workflow systems can support task migration to other resources using checkpointing.

Based on these taxonomies Wieczorek et al. examine the existing approaches on Grid workflow scheduling. Their classification of general-purpose Grid workflow systems is given in Table 3.1. In the survey the GWES is listed under the designation Instant-Grid [54]. MediGRID uses a newer version of GWES, however its (previous) scheduler matches the scheduling in Instant-Grid regarding the taxonomies.

With regard to our work the dynamism of the scheduling process is of special importance. Several approaches try to address the trade-off between optimization of the overall workflow execution which requires a look-ahead planning, and the ability to dynamically react on changes in the Grid. The Vienna Grid Environment (VGE) [91] supports full-ahead and just-in-time scheduling but either one or the other is employed for scheduling. The hybrid approaches that combine full-ahead and just-in-time scheduling can be separated into two categories. The first class is based on partitioning the complete workflow into smaller subworkflows. These subworkflows are then scheduled just-in-time using a full-ahead approach. A representative of this class is PEGASUS [93, 94].

The second class performs full-ahead scheduling of the complete workflow Graph which is repeated whenever task executions fail or the state of the Grid changes. Rescheduling is used by the Nimrod/G resource broker [88, 104] and the GrADS system [89, 90] as well as in the theoretical approach of Yu and Shi [105] which employs HEFT for full-ahead scheduling.

Qin et al. [106] propose a new architecture for ASKALON [95, 96, 97, 98] which replaces the HEFT-based full-ahead scheduling by a just-in-time scheduler that prioritizes tasks according to their the distance to the end of the workflow. However, no evaluation of the architecture is given.

In their final analysis Wieczorek et al. [87] point out the relevance of hybrid approaches by stating: “Finally, we think that a hybrid scheduling model is the most appropriate for multi-criteria workflow scheduling in a dynamic and heterogeneous Grid environment. Just-in-time models can be really successful when optimizing execution time, and full-ahead scheduling would work best for

Table 3.1: Survey of general-purpose Grid workflow systems [87]

Paper/System		Nimrod/G [88]	GrADS [89, 90]	VGE [91, 92]	PEGASUS [93, 94]	ASKALON [95, 96, 97, 98]	K-WfGrid [99, 52, 100]	Instant-Grid [54]	P-Grade [101]	UNICORE6/Chemomentum [102]	Knowledge Grid [103]
Workflow model	task oriented	+	+	+	+	+	+	+	+	+	+
	task and data oriented	'	-	-	-	-	-	-	-	-	-
	DAG	'	+	+	+	-	-	-	+	+	+
	extended digraph	'	-	-	-	+	+	+	-	-	-
	simplified DAG	+	-	-	-	-	-	-	+	-	-
	tunable workflows	'	'	+	+	'	+	-	-	-	-
	single input workflows	+	+	+	+	+	+	+	+	+	+
	pipelined workflows	'	-	-	-	-	-	-	+	-	-
Criteria	multiple criteria	+	'	+	'	+	+	+	+	+	+
	execution time	+	+	+	+	+	+	-	+	+	+
	execution cost	+	-	-	-	-	-	-	-	-	-
	reliability	'	-	-	-	-	-	-	-	-	-
	data quality	'	-	-	-	-	-	-	-	-	-
	generic criteria model	'	-	+	-	+	+	-	-	-	+
	Grid-oriented criteria	'	-	-	-	+	-	+	+	+	-
	adaptive cost model	+	'	-	-	-	-	-	-	-	-
Scheduling process	multiple workflows	'	'	-	+	+	-	+	+	-	-
	just-in-time	'	'	+	'	'	-	+	+	+	-
	full-ahead	'	+	+	+	+	+	-	-	-	+
	hybrid	+	+	-	+	-	-	-	-	-	-
	advance reservation	'	-	+	-	+	-	-	-	-	-
Resour.	heterogeneous resources	+	+	+	+	+	+	+	+	+	+
	multiprogrammed resources	'	'	-	-	-	-	+	-	-	-
Task	moldable tasks	'	+	+	+	-	-	-	-	-	-
	malleable tasks	'	+	-	-	-	-	-	-	-	-
	migrative tasks	'	+	-	-	-	-	-	-	-	-

the other criteria; assuming the dynamic and unpredictable nature of the Grid, a solution that combines the advantages of these two models would provide best expected results." This corresponds to our opinion and describes what we tried to achieve when we developed the two-tier approach which is presented in Chapter 5.

Chapter 4

Queue Waiting Time Prediction

4.1 Measurement program

To investigate the actual and time-varying availability of resources at D-Grid computing sites, we conducted a number of experiments. For this purpose, we developed a measurement program and installed it on the head nodes of 11 large clusters each located at a different computing center. These clusters comprise in total more than 16000 processor cores. Upon invocation, the measurement program submits a test job. The job characteristics are identical at all sites to allow for comparison. The test jobs are serial jobs, i.e. they require one processor core. The requested execution time was set to 4 hours and 1 minute which means that the LRMS has to find a time slot for the job in its schedule that offers at least this duration. The requested execution time of a job is also denoted as the walltime. We derived this walltime from statistical analysis of the average execution time of MediGRID jobs at the site GWDG which was approximately 4 hours. The specification of the walltime is necessary because LRMS are typically configured with several queues for different jobs lengths where shorter queues have higher priorities. This way, the walltime influences the queue waiting time of the jobs. With our specification we avoid so-called express queues and send our test jobs to production queues which allow at least 4 hours job execution time. In the same time, this makes the measurement program site-independent, because we do not have to specify a site-specific queue name in job submission. In D-Grid there are no general conventions on queues so that many sites use different queue names

or configure different properties for queues with identical name. By specifying the walltime, jobs are routed by the LRMS to the appropriate queue. The employed test jobs are so-called "empty jobs" which means that their actual processing time is only a few milliseconds. This is because they should not unnecessarily occupy compute resources. The measurement program is executed by means of a regular MediGRID user account. This account is used solely for testing and does not run production jobs that consume a notable amount of resources.

The measurement program is run periodically, once per hour on every site. It is triggered via the Unix cron daemon or a start script if cron is not available. The hourly interval was chosen not to disturb production use. On every invocation it retrieves a timestamp of the current system time, the total number of running jobs at the site, the total number of waiting jobs at the site, the job identifier of the submitted test job, and the turnaround time of the submitted test job. [1] As test jobs are empty, the turnaround time approximately equals their queue waiting time. All these values are written on one line into an output file. The first four parameters except the turnaround time are output instantly, and after the test job has finished the preliminary output is replaced by a new line that contains the turnaround time. For this, the output file must be locked to prevent multiple concurrent writes to the file.

During inspection of the output file we noticed a number of problems. First of all, output lines can be missing or incomplete due to problems with the LRMS or downtimes of the resource. Furthermore, also multiple output lines occurred during rare disturbances on a cluster. A surprising discovery was to see that often output lines were not in the supposed chronological order. This situation can occur when multiple test jobs from several measurement intervals have been queued for hours and all of them finally get executed at the same time on different processors in the cluster. We created another program that corrects all of the aforementioned errors. It sorts all output lines chronologically, removes duplicate outputs of the same measurement interval, and inserts lines for missing output which contain just the respective timestamp. In the end, the output file contains one output line for every hour between the first and the last invocation of the measurement program.

[1] In the following, we use the term turnaround time to denote the overall completion time of jobs. It consists of the processing time and (potential) delays due to overheads or waiting times.

4.2 Measurements in D-Grid

As a result of our measurements, we have discovered four intrinsic problems for meta-scheduling that limit the possibilities of Grid meta-schedulers. Comparable findings are also described in [79].

These problems are:

1. Compute resources used in Grid computing typically consist of large clusters that are used by several Grid projects and multiple VOs. Each of these VOs can use its own Grid scheduler. In this case, each Grid scheduler only has the role of a "power user" that competes with other users, e.g. other Grid schedulers in the same Grid from the perspective of the LRMS. As a consequence, an optimal schedule is not possible for any individual meta-scheduler. We call this the competing-schedulers problem.

2. Computing sites that take part in Grid projects retain the concept of site autonomy. This means that administrative decisions and privileges remain at the site level. Therefore, Grid schedulers have no control over the site scheduler's policy, and over the prioritization of the jobs waiting in the queues. Thus, no control exists over the respective LRMS for the meta-scheduler. We call this the lack-of-control problem.

3. Every site uses a custom configuration of queues, and processors can be shared among queues or can be dedicated to queues exclusively. Usually the information from the LRMS does not allow to reliably determine the number of available processors. The only statistics commonly available are the number of running and waiting jobs. Some sites do not even provide this information because of nondisclosure agreements. Additionally, sites employ custom configurations for their local scheduling which are usually non-transparent to the users. To conclude, local schedulers currently do not provide sufficient information for a good schedule at the Grid level. We call this the information-insufficiency problem.

4. Resources are normally highly utilized, and waiting times at clusters can last up to hours. Therefore, input queue waiting time can considerably exceed the actual processing time for small jobs and is their dominant factor.

> However, input queue waiting time and input queue length have shown to be not continuously differentiable functions over time. Instead, they can vary within minutes by a factor of thousand or more. They behave more like fractals than functions. This makes predictions of future queue waiting times and queue lengths a delicate task. However, scheduling always relies on such predictions. We call this the non-continuously differentiable-function problem.

In the following we present details of the measurements on all 11 D-Grid sites. For reasons of clarity the graphs only show results from a measurement timeframe which was 720 hours (30 days). Figs. 4.1-4.11 show the changes of waiting time, running jobs and waiting jobs. The left y-axis denotes the waiting time in minutes, the right y-axis denotes the number of waiting and running jobs.

The queue waiting time is dependent on the scheduling of the test jobs by the site scheduler that is part of the LRMS. From site to site, there are a number of typical scheduling criteria: Jobs can be assigned static priorities based on e.g. user, group and job class. Fairshare-scheduling prioritizes jobs based on historical resource usage. It relates recorded resource usage to utilization targets for e.g. users, groups and job classes. Furthermore, jobs can be prioritized depending on the amount of requested resources. A traditional and common method is to raise priority with increasing input queue waiting time. Many sites employ a combination of several criteria. Depending on the configuration of the site scheduler, other users such as non-members of MediGRID might have experienced greatly different waiting times than our diagrams show.

At site 1 (see Fig. 4.1), we can see big differences in waiting times. Most of the time the test jobs were executed instantly. Otherwise steep peaks occurred which reach up to almost five hours and also last for this time. The number of waiting jobs is gradually decreasing during most of the time interval. There is no correlation between waiting time and the number of waiting jobs except for one peak at the 520th hour in the interval. This means test jobs were often executed instantly even though many other jobs were already waiting in the queue. The number of running jobs is low and changes several times. This must be caused by changes in the parallelism of jobs as there is a high number of queued jobs all the time. Because of the missing correlation between waiting time and number of waiting jobs, the local scheduling of site 1 is probably based on a fairshare

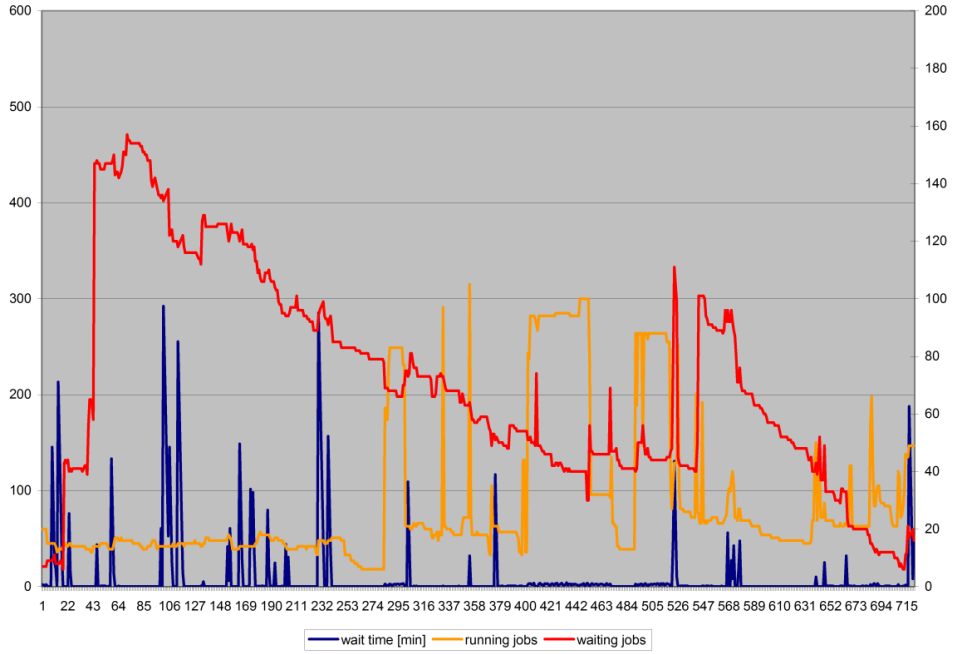

Figure 4.1: Variation of input queue waiting time and number of jobs in 720 hs at site 1.

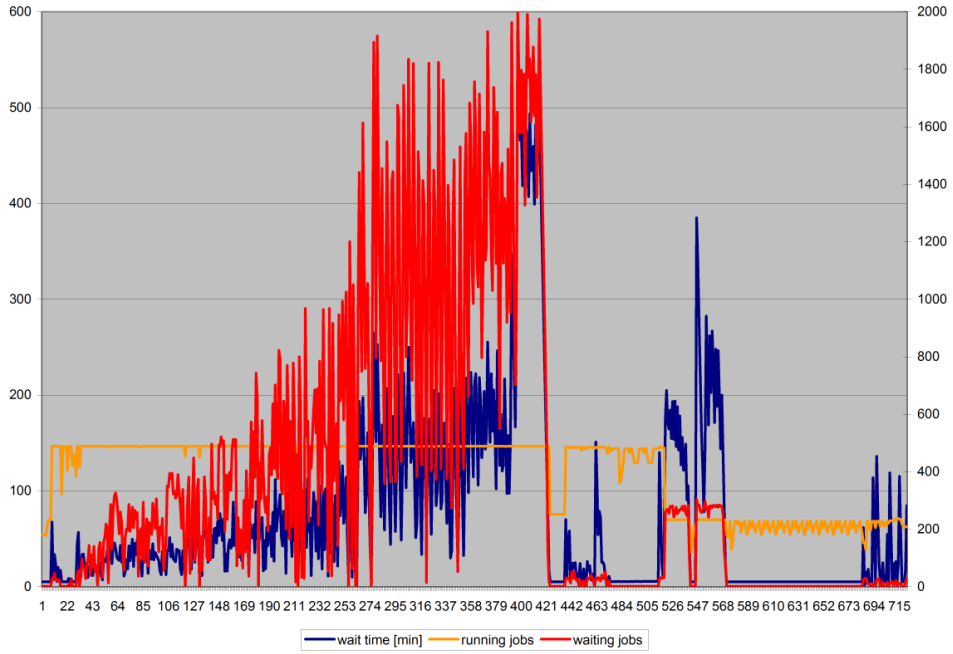

Figure 4.2: Variation of input queue waiting time and number of jobs in 720 hs at site 2.

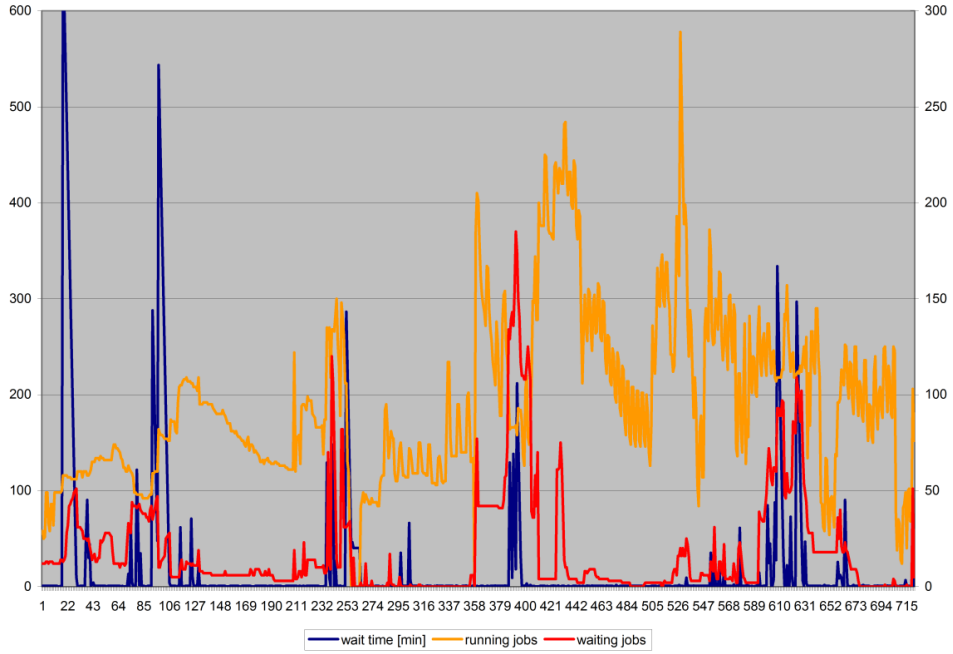

Figure 4.3: Variation of input queue waiting time and number of jobs in 720 hs at site 3.

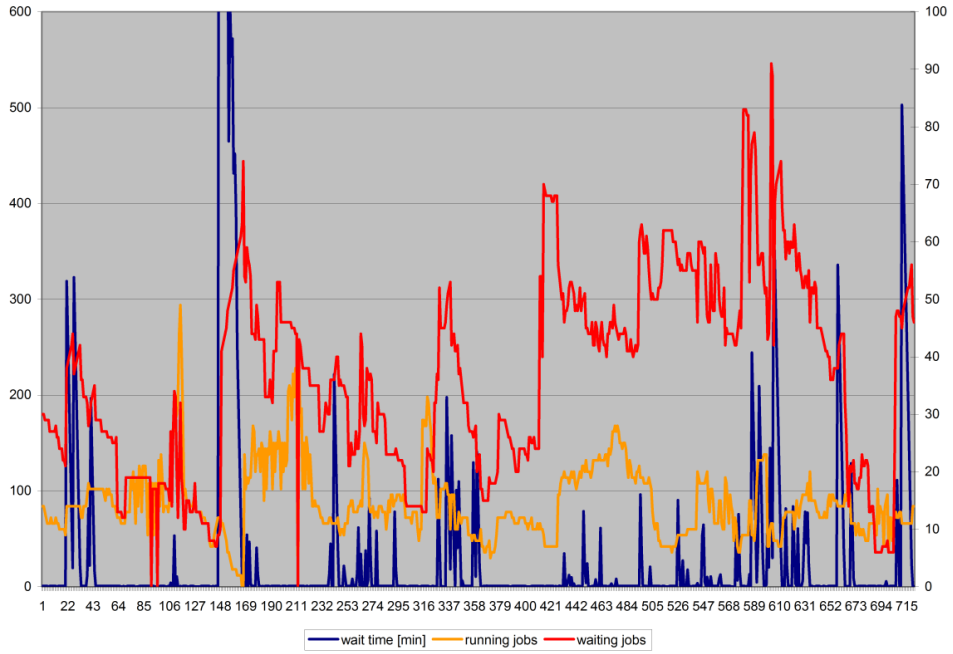

Figure 4.4: Variation of input queue waiting time and number of jobs in 720 hs at site 4.

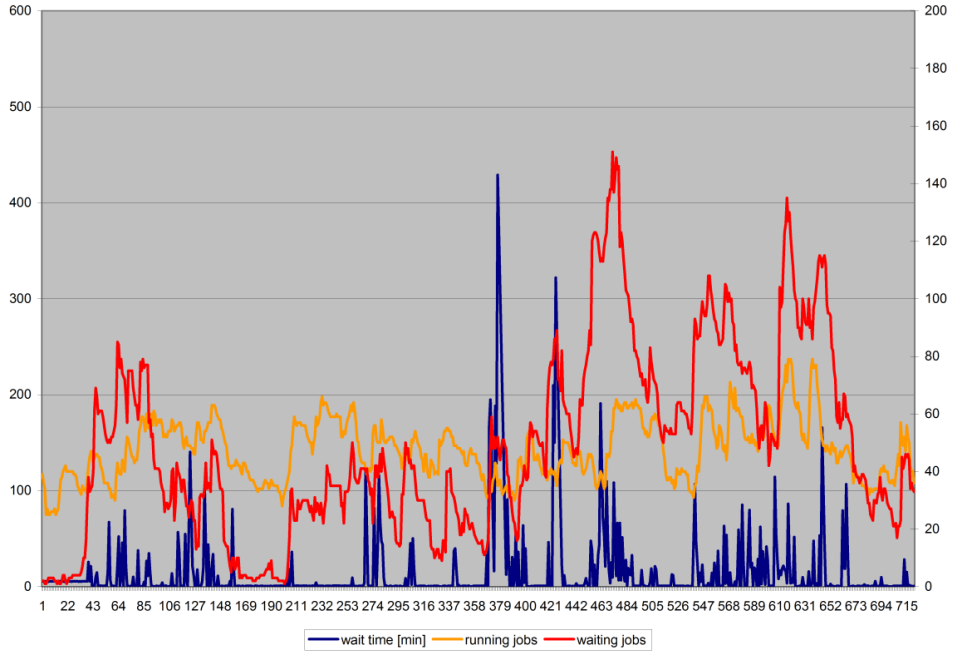

Figure 4.5: Variation of input queue waiting time and number of jobs in 720 hs at site 5.

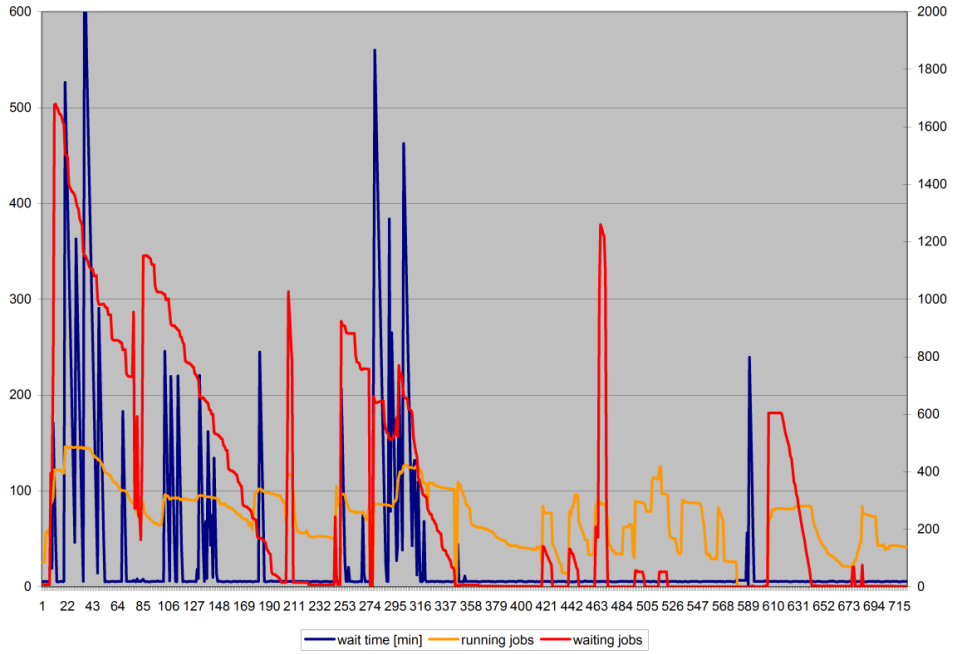

Figure 4.6: Variation of input queue waiting time and number of jobs in 720 hs at site 6.

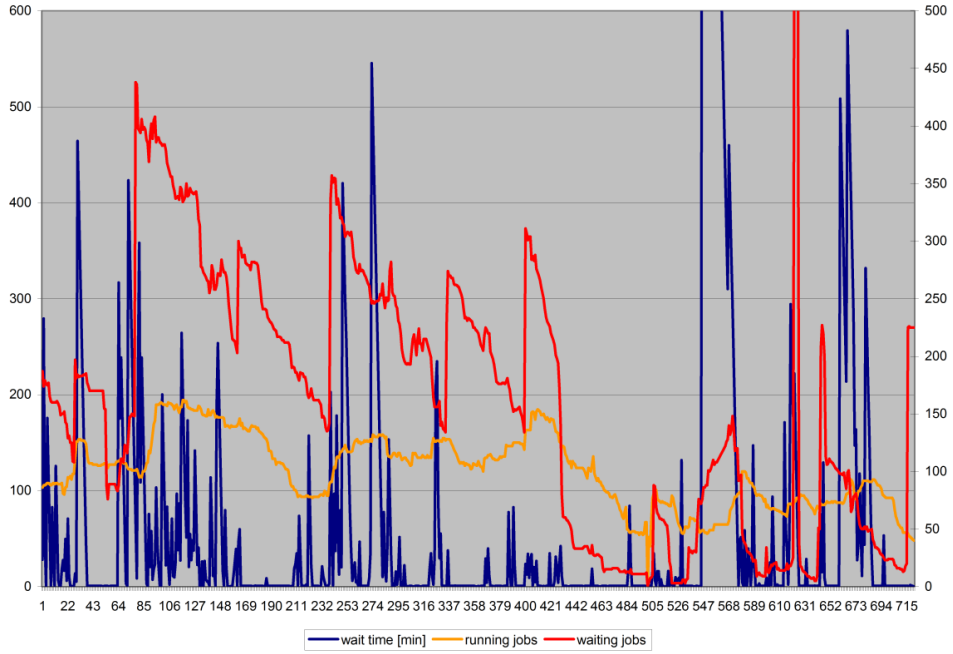

Figure 4.7: Variation of input queue waiting time and number of jobs in 720 hs at site 7.

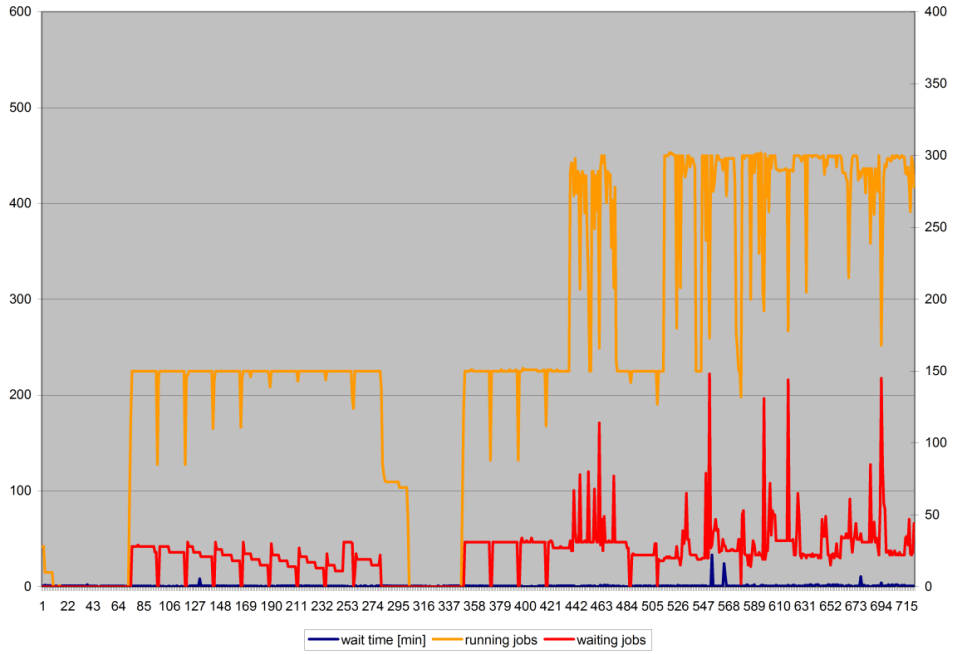

Figure 4.8: Variation of input queue waiting time and number of jobs in 720 hs at site 8.

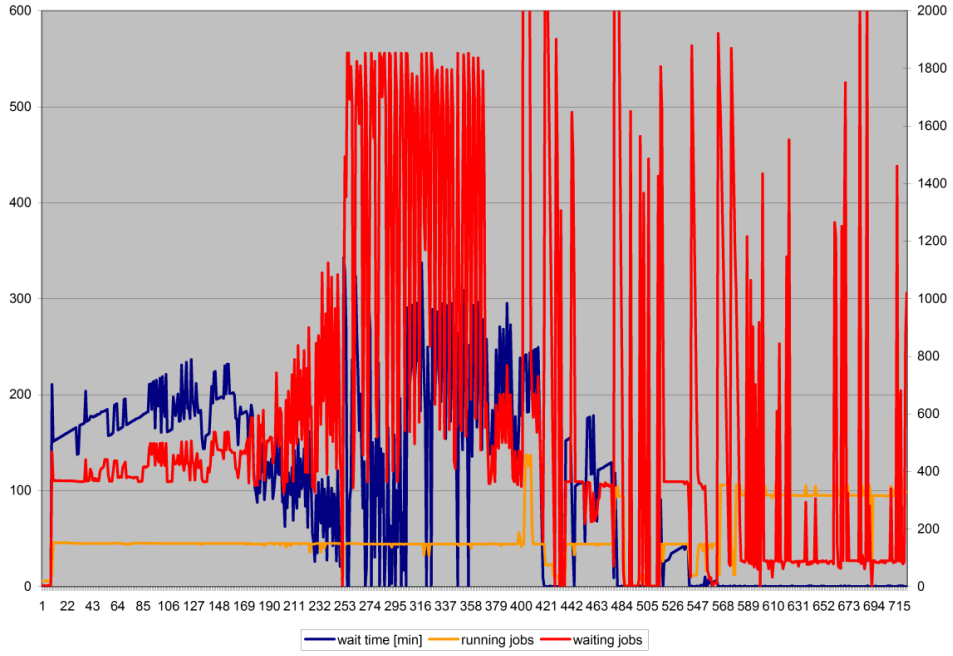

Figure 4.9: Variation of input queue waiting time and number of jobs in 720 hs at site 9.

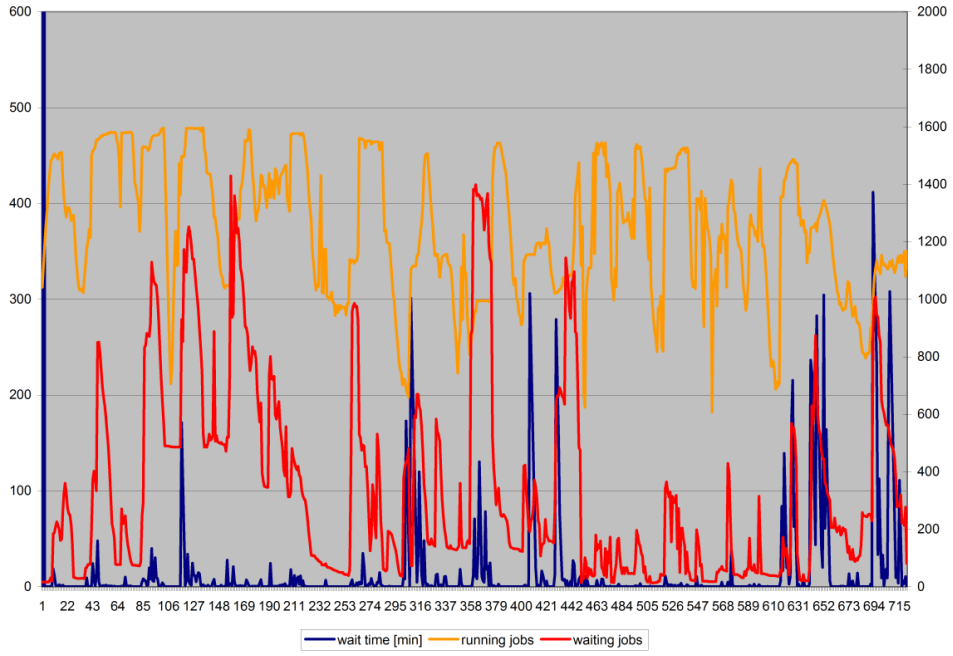

Figure 4.10: Variation of input queue waiting time and number of jobs in 720 hs at site 10.

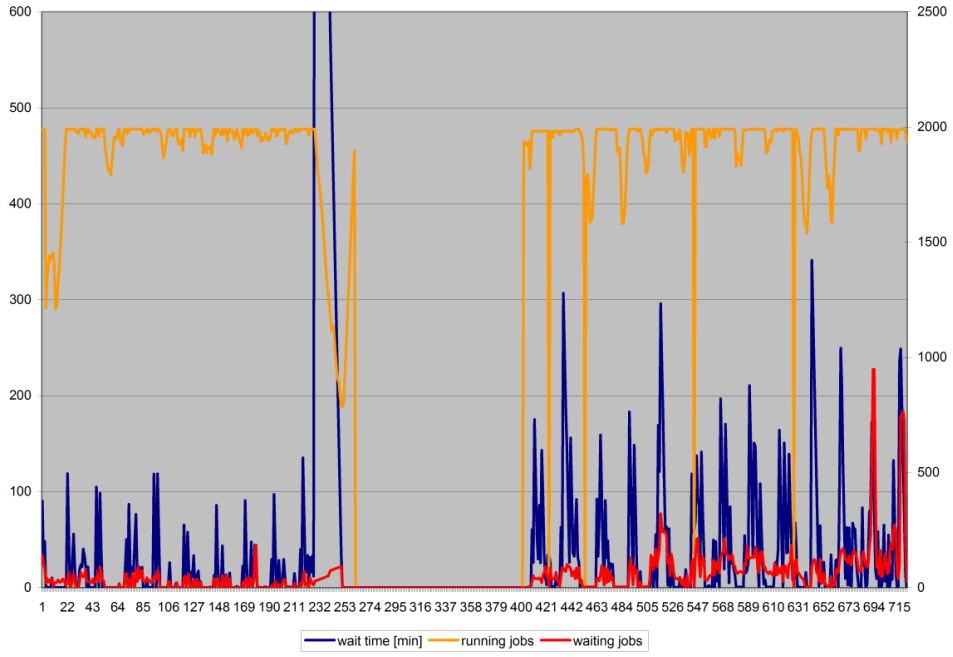

Figure 4.11: Variation of input queue waiting time and number of jobs in 720 hs at site 11.

algorithm.

The graphs of site 2 (see Fig. 4.2) show an increasing waiting time, together with an increase in the number of waiting jobs. At the peak, waiting time is almost ten hours, then both values settle down to approximately zero. The second half of the interval is characterized by peaks in waiting time that still correlate with the number of waiting jobs. The number of running jobs remains constant during most of the interval, this is typical if jobs are homogeneous. The strong correlation between waiting time and waiting jobs leads to the conclusion that site 2 prioritizes jobs based on the input queue waiting time, i.e. with the first-come first-served (FCFS) queueing discipline.

At site 3 (see Fig. 4.3), the test jobs were either executed with no waiting time or long waiting times of even more than ten hours. The graphs show a dependency between waiting time and number of waiting jobs during most of the peaks. The number of running jobs varies a lot and is usually much higher than the number of waiting jobs. This can be an indication that users or meta-schedulers addressing the site try to avoid further submission when many jobs

are already queued. Therefore, the number of running jobs seems to be more dependent on the amount of submissions than job parallelism. At the same time the cluster was probably often not fully utilized during the interval. Due to the dependency between waiting time and waiting jobs the scheduling of site 3 certainly incorporates the FCFS principle.

The waiting times at site 4 (see Fig. 4.4) differ a lot. It shows phases with no waiting time that are interrupted by peaks. The highest peak takes twenty hours until queue waiting time settles down again. There is no continuous correlation between waiting time and number of waiting jobs. The number of running jobs is low, so this site probably runs a lot of parallel jobs that occupy many processor cores. At some points in time different types of jobs seem to arrive at the site which have a similar prioritization as MediGRID jobs and lead to a prolongation of measured waiting times. However, overall the scheduling is probably based on a fairshare method.

At site 5 (see Fig. 4.5), waiting times vary a lot. There are frequent peaks that last one to two hours. Higher peaks only occur rarely. The graphs show some dependency between the peaks and the number of waiting jobs but there is no continuous correlation with waiting time. The number of running jobs fluctuates around 50. This probably originates from the site running a mix of bigger and smaller jobs in terms of used processor cores. The scheduling of site 5 is probably dominated by the fairshare principle.

The graphs of site 6 (see Fig. 4.6) are characterized by many peaks in waiting time which are very high with one peak exceeding 10 hours. A dependency between waiting time and waiting jobs does not exist. It is noticeable that the number of waiting jobs often increases by several hundred then settles down at a rate of roughly 10 jobs per hour. This means that job submission to the site occurs in large batches. The number of running jobs varies. In the first half of the interval this might be related to a mixture of jobs regarding parallelism. In the second half of the interval the site was certainly not fully utilized. The scheduling of site 6 is probably based on a fairshare algorithm.

At site 7 (see Fig. 4.7), there are frequent peaks in waiting time many of which are very high and last more than 5 hours. The highest peak is more than 24 hours. There is no dependency between waiting time and the number of waiting jobs. The number of running jobs varies slowly but is lower in the second half of the

interval. Job prioritization at site 7 is probably based on fairshare consideration.

At site 8 (see Fig. 4.8), the test jobs almost never had to wait before execution. At the same time, there is no dependency visible between waiting time and the number of waiting jobs. The test jobs were executed instantly in spite of the other jobs already waiting in the queue. The maximum number of running jobs increases during the interval. This means that either the number of processor cores was increased or fewer cores were used by jobs submitted later. The local scheduling of site 1 probably uses the fairshare method or it assigns very high static priorities to MediGRID jobs.

During the first seven days of the presented interval, waiting times at site 9 (see Fig. 4.9) lasted on average three hours and correlated with the number of waiting jobs. Afterwards, both values show high fluctuation and no continuous correlation. In the last seven days, the test jobs experienced almost no waiting time even though other jobs were waiting in the input queue. The number of running jobs is very constant with a step at the beginning of the last week. The scheduling at site 4 appears to use mainly FCFS, except from the last week. Then, it is changed to fairshare or high prioritization for MediGRID jobs.

The graphs of site 10 (see Fig. 4.10) show immense variations in queue waiting times. Long periods with very low waiting times are interrupted by steep peaks which last for a few hours. A dependency between waiting times and waiting jobs only seems to exist during the last peaks in the observed interval. This might be an indicator that these jobs received a similar prioritization as the test jobs. The number of running jobs varies a lot while jobs are still queued which points to changes in the parallelism of jobs. The scheduling of site 10 is certainly fairshare-based.

At site 11 (see Fig. 4.11), the execution of the measurement program was interrupted for a few days, therefore there is a period without data in the center of the 30 day interval. The waiting times show a very clear periodicity of about 24 hours. Additionally, a clear correlation can be observed between waiting time and waiting jobs. The number of running jobs is almost constant, i.e. jobs are very homogeneous and probably serial jobs. The correlation between waiting time and waiting jobs leads to the assumption that site 11 uses FCFS for job prioritization.

The graphs of the 11 D-Grid sites have shown that at most clusters the input queue waiting times vary a lot in magnitude. Periods where the test jobs are

executed instantly or with little waiting time alternate with periods of very high waiting times. The transition from one period to the other does not take place gradually. Instead the value of waiting time often jumps from one measurement to the other. To examine whether the reason behind this observation is just the coarse measurement interval of hourly measurements, we performed further tests. At five sites additional measurements were executed with a shortened submission interval of the test jobs that was only 5 minutes instead of 1 hour. These tests were run for 14 days and then stopped not to overly disturb normal cluster operation. The analysis of the data showed very similar results as with the hourly measurements. In particular the series of waiting times still contained high jumps where waiting time jumped from approximately zero to several hours from one measurement to the other. This shows that input queue waiting time is discontinuous and that the curve shape in Figs. 4.1-4.11 is not caused by limited measurement.

4.3 Estimation Methods

The measurements on D-Grid clusters have demonstrated that their availability is determined by the input queue waiting times. An effective meta-scheduling across multiple clusters therefore depends on information about the expected queue waiting times. This is supported by the fact that waiting times can exceed job execution times and are characterized by abrupt changes. However, as we noted in our description of the information-insufficiency problem, the local schedulers in current LRMS do not provide this information. For this reason we decided to develop an own methodology to predict queue waiting times of D-Grid clusters. The primary objective of this methodology is to detect the beginning of peaks because in this case no jobs should be submitted to the site anymore. The prediction of the actual duration becomes more important the more sites have high utilization.

Our approach is to calculate an estimated value for the waiting time of a new job entering the queue based on the data that is continuously acquired by the measurement program. This data comprises the number of running jobs at the site, the number of waiting jobs, the job identifiers of the test jobs, and their measured queue waiting times.

During the analysis of collected site data we identified three classes of recurring characteristics depending on the local scheduling employed at the site. There are sites with no correlation between waiting time and the number of waiting jobs. These sites probably use a fairshare algorithm or assign high static priorities to MediGRID jobs. Furthermore, there are sites where waiting time correlates with the number of waiting jobs probably because prioritization follows the first-come first-served principle. Finally, we found periodicity in waiting time series which can occur with both scheduling conditions.

To reflect these different characteristics, we propose a prediction methodology that combines three alternative estimation methods s, u and v that are based on three different concepts in order to reflect the three classes we have divided the measurements into. For all resources, the performance of the three estimations is evaluated and that estimation is used which performed best in the latest prediction cycles, i.e. the predicted value will be either s, or u, or v depending on the accuracy of the previous estimations.

The first estimation, s, uses the method of exponential smoothing to estimate the next value of queue waiting time based on previous values. Exponential smoothing is equivalent to a first-order infinite impulse-response low-pass filter [109]. The estimation value is calculated as $s_t = \beta * n + (1 - \beta) * s_{t-1}$, where n denotes the latest measured queue waiting time, and s_{t-1} is the previous estimated value of s. $\beta \in [0,1]$ is called the smoothing factor. After evaluation by comparison with measured data (cf. Section 4.6) we set $\beta = 0.5$ which results in moderate smoothing and emphasizing of peaks in the series of waiting times because peaks appear frequently. As a possible improvement, multiple values of s could be calculated for different requested job walltimes because some sites have exclusive nodes for short and long running jobs.

The second estimation, u, uses Little's law [110] from queueing theory to calculate the expected value measure of queue waiting time. Application of Little's law requires to determine two basic parameters of the queueing system, the average rate of jobs entering the queueing system λ, and the average rate of serving jobs μ. As mentioned before, meta-schedulers do not have insight into the LRMS and current LRMS do not provide these values. Therefore, we have developed a procedure to determine λ and μ that requires user permissions on the site only.

The measurement program which is installed on each cluster periodically retrieves

the number of running and waiting jobs and the identifier of the test job. The LRMS assigns job identifiers that contain consecutive numerical values. Therefore, submitting a job and immediately afterwards reading its identifier allows to reliably determine the job number of the latest job. The difference between the values of the current and the previous measurements gives the arrival rate λ of jobs in the cluster during the submission time interval. The total service rate of the cluster μ' can be calculated by adding the number of jobs that have arrived during the time interval to the total number of jobs having been previously in the system, and by subtracting the total number of jobs currently in the system.

$$\begin{aligned} \lambda &= job\_number_t - job\_number_{t-1} \\ \mu' &= run_{t-1} + wait_{t-1} + \lambda - (run_t + wait_t) \end{aligned}$$

Now we can apply Little's law which relates the steady-state mean system-sizes to the steady-state average waiting-times. A steady-state denotes the situation of the system after it has been settled, i.e. after running for a long time. Prerequisite for reaching a steady-state is that $\rho = \lambda/c\mu = \lambda/\mu'$ (often called traffic intensity) must be strictly less than 1. c is the number of parallel servers in the cluster. When $\rho > 1$ the queue size never settles down and there is no steady state possible.

According to [110] we define N as the total number of jobs in the system (running or waiting), N_q as the number of jobs waiting in queue, T as the total time a jobs spends in the system, and T_q as the time a job spends waiting in the queue. Then, we define the expected value measures of these terms. $L = E[N]$ is the mean number of jobs in the system, $L_q = E[N_q]$ is the mean number of jobs waiting in queue, $W = E[T]$ is the mean waiting time in the system, and $W_q = E[T_q]$ is the mean waiting time in queue. Little's laws for general queueing systems (G/G/c) are $L = \lambda W$ and $L_q = \lambda W_q$. Thus, we get the expected queue waiting time as $u = W_q = L_q/\lambda$. In the practical evaluation it turned out to be advantageous to use the current queue length for L_q, and to average λ over the last 12 hours.

The third estimation, v, is based on a Fourier analysis [111, 112] of the time series of measured input-queue waiting-times. We perform a Discrete Fourier transform (DFT) that decomposes the sequence of values over time into components of different frequencies. In the frequency domain, the sinusoidal basis functions of the decomposition are treated by us with respect to their amplitudes. Only high

amplitudes are preserved. Sinus oscillations with low amplitudes are ignored. Clipping low coefficients results in a filter operation. Afterwards, the remaining coefficients are transformed back into the time domain using the inverse DFT. Since the DFT only generates frequency components needed to reconstruct the finite time segment that was analyzed, its inverse transform cannot reproduce the entire time domain, unless the input happens to be periodic. For this estimation, we assume that the input data is periodic. This holds true for some sites because there are Grid jobs that are submitted automatically at fixed points in time around-the-clock. Thus we obtain the estimated value v as the first value of the periodic extension of the resulting output sequence. The modification of amplitudes in frequency domain is made to emphasize the periodicity of the sequence. The DFT is applied to the interval of the last 128 values of the waiting time series. With hourly measurements this means that roughly five days are considered for periodicity.

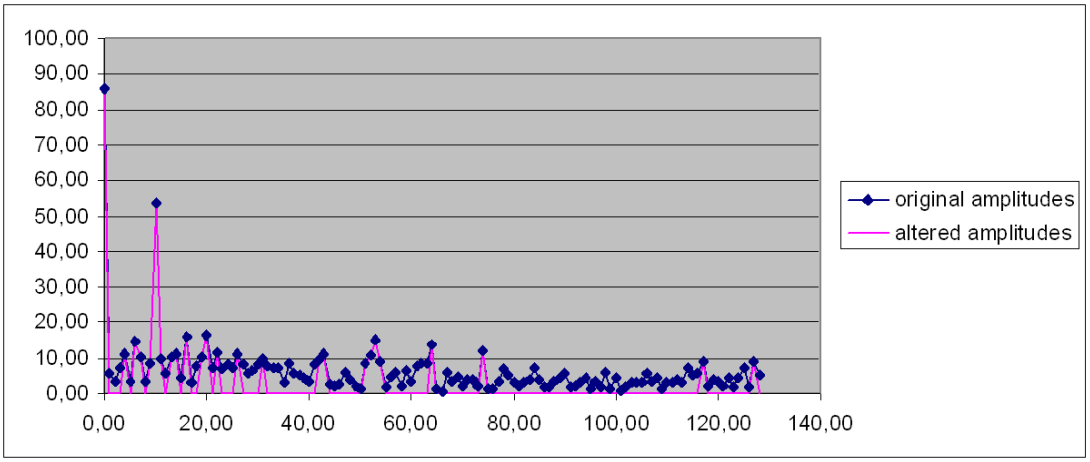

Figure 4.12: Amplitudes of a sequence of waiting times from site 11.

An example for DFT data is shown in Figs. 4.12 and 4.13. This Fourier analysis was done using Excel [113, 114]. In Fig. 4.12, the amplitudes of a sequence of waiting times of site 11 are shown with original amplitudes as well as with clipped values. The periodicity of the time series is clearly noticeable from the dominant peak in the frequency spectrum that appears at $f > 0$. The peak occurs at the tenth sinusoidal basis-function frequency because we have sampled ten periods in the original time sequence. All other frequencies (except $f = 0$ which denotes the irrelevant DC part of the signal) have much smaller amplitudes in the spectrum.

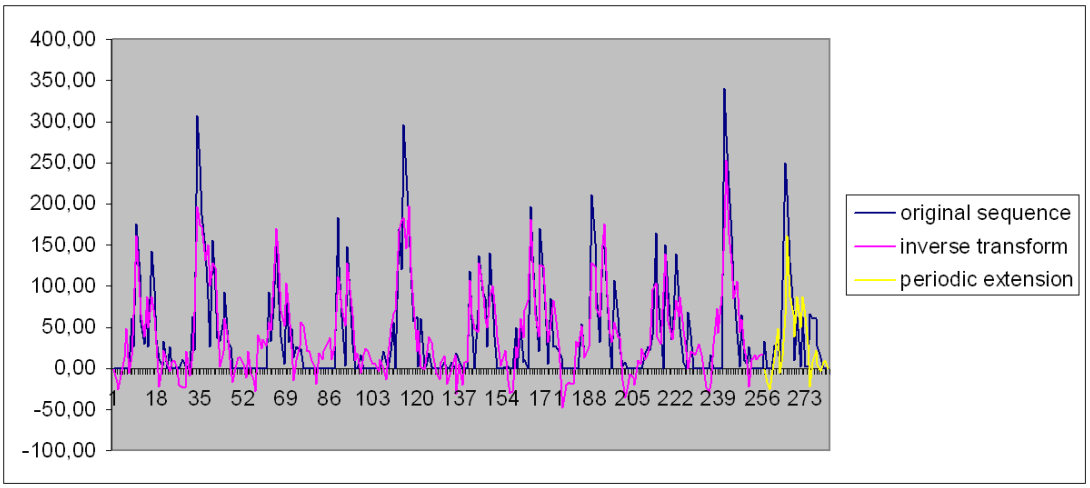

Figure 4.13: Original sequence of waiting times from site 11 and retransformed spectrum into the time domain of the modified, i.e. clipped spectrum.

After having obtained the amplitude part of the spectrum, we applied a clipping procedure as follows to the amplitudes: If the magnitude is less or equal than 10% of the maximum amplitude the respective amplitude is set to zero. The periodic extension of the retransformed spectrum shown in Fig. 4.13 resembles the original continuation very well. This means that DFT is a good extrapolation method for this class of sites. It should be pointed out again that only the first yellow point is used from one DFT calculation. The successive values in time are calculated by shifting the input window one step right and repeating the calculation. As can be seen in the graph, the inverse transform can contain negative values, i.e. queue waiting times. Those are set to zero.

4.4 Evaluation of Estimation Methods

To evaluate the three estimation approaches, they have been implemented in a Java program and applied to the data acquired by the hourly measurements at the 11 D-Grid sites. The program calculates the predictions of exponential smoothing, Little's law and Fourier analysis for every hour in the provided input range. Afterwards, all three estimation values are compared to the respective measured value. This allows for automated retrospective assessment of the esti-

mation methods regarding their deviation from the actual queue waiting times.

In the following we present a graphical evaluation of the predictions for all 11 D-Grid sites. We show the results for the same timeframe of 720 hours (30 days) which was introduced in Section 4.2. Figs. 4.14-4.24 show the predictions based on exponential smoothing, queueing theory and Fourier analysis in comparison to the measured queue waiting times where the y-axis denotes the time in minutes.

For site 1 (see Fig. 4.14), estimation 1 (exponential smoothing) performs reasonable. However, the peaks are predicted one time step later than in the original series which is inevitable to first-order filtering. Estimation 2 (Little's law) performs very bad and is not applicable to site 1. Estimation 3 (Fourier analysis) correctly predicts the peak at the 229th hour in the interval and reveals that there is some periodicity in the sequence. The advantage of estimation 3 is that it can predict peaks without delay. The downside is that it falsely predicts peaks or misses peaks if the waiting times are not continuously periodic.

For site 2 (see Fig. 4.15), estimation 1 performs well. This site illustrates the importance of the smoothing factor β for the suppression of discontinuities in the queue waiting times. Estimation 2 also performs well, only a few peaks are not predicted properly. Its predictions are often better than those of exponential smoothing as it does not have a delay as estimation 1. This is especially noticeable at the end of the dominant peak in the diagram. Estimation 3 works as long as the trend of waiting times changes only gradually. In the second half of the interval that features nonperiodic peaks, estimation 3 fails to provide useful predictions.

For site 3 (see Fig. 4.16), estimation 1 performs reasonable. The main issue is again the delay in peak detection. Additionally, after the peaks the prediction does not fade instantly. This could be improved by choosing a greater smoothing factor, but that would also imply more jitter which is not desirable. Estimation 2 mostly overestimates waiting times in the first half of the interval and still misses some of the peaks. In the second half it still shows many falsely predicted peaks but detects all peaks prior to their beginning. Estimation 3 is not applicable to site 3.

For site 4 (see Fig. 4.17), estimation 1 performs reasonable. It is noticeable that exponential smoothing is usually unable to predict the exact height of peaks. However, this is a secondary issue because as long as the presence of a peak is detected, no jobs are sent to the site anymore. Estimation 2 does not deliver a

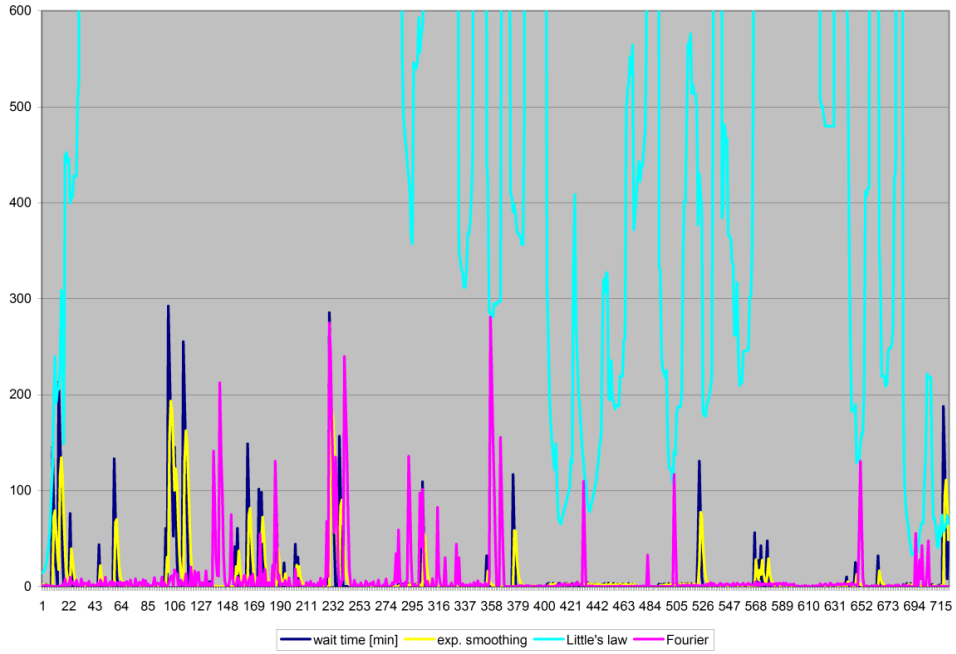

Figure 4.14: Comparison of measured input queue waiting time and estimations for site 1.

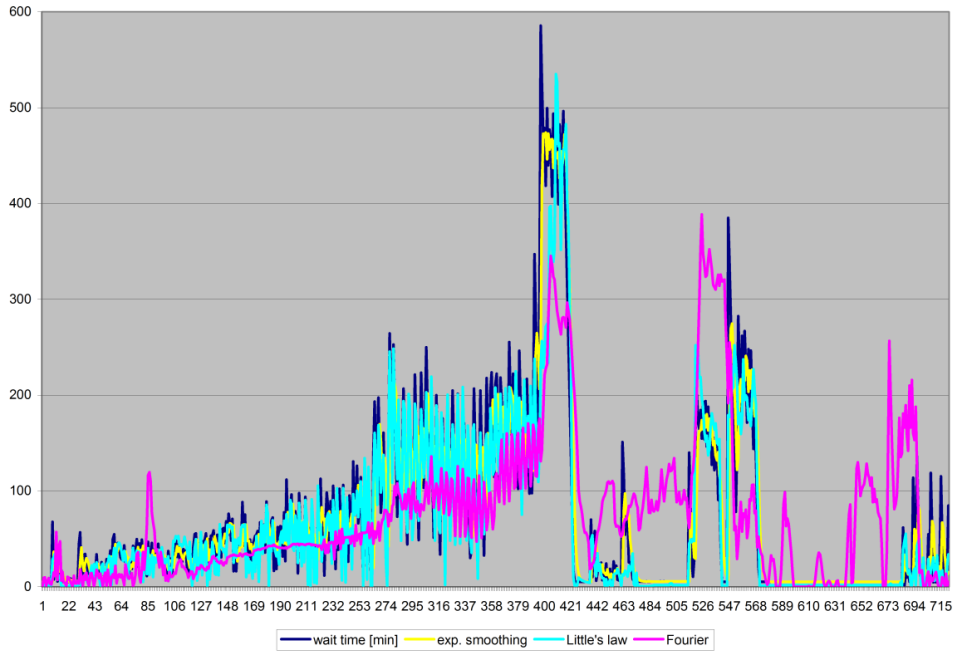

Figure 4.15: Comparison of measured input queue waiting time and estimations for site 2.

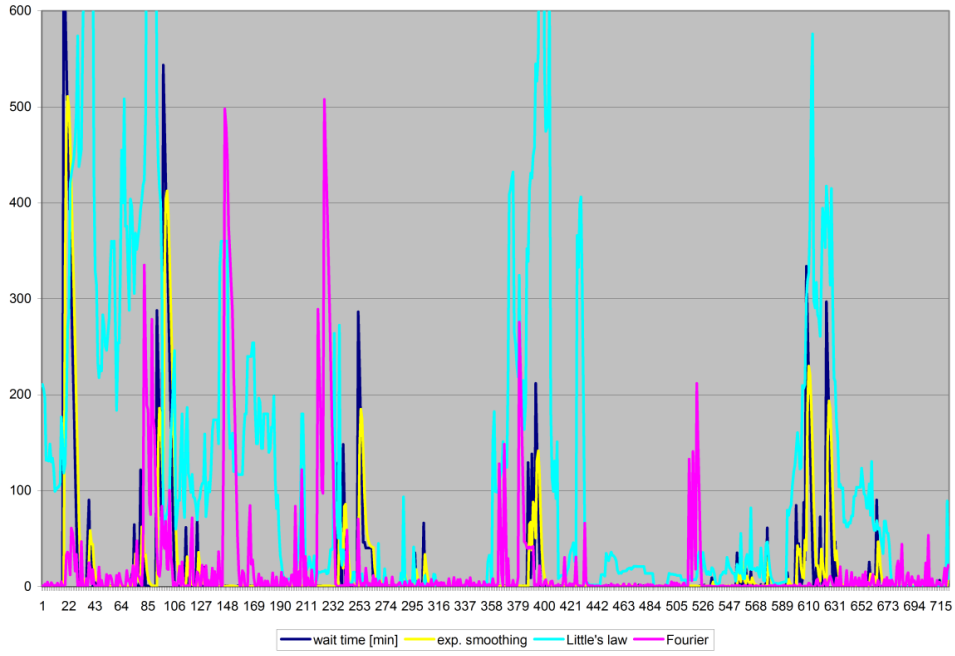

Figure 4.16: Comparison of measured input queue waiting time and estimations for site 3.

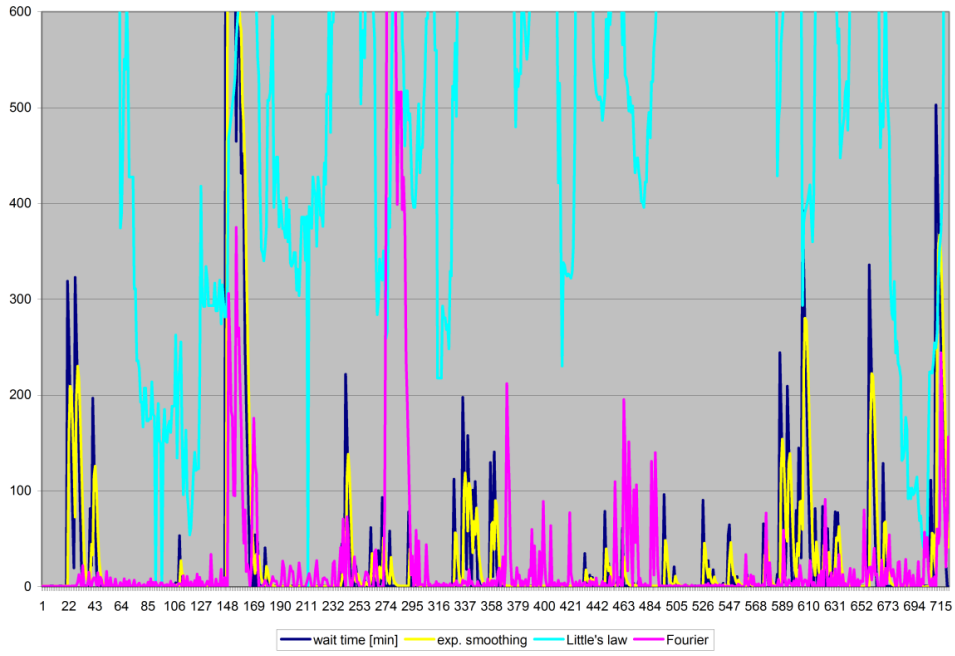

Figure 4.17: Comparison of measured input queue waiting time and estimations for site 4.

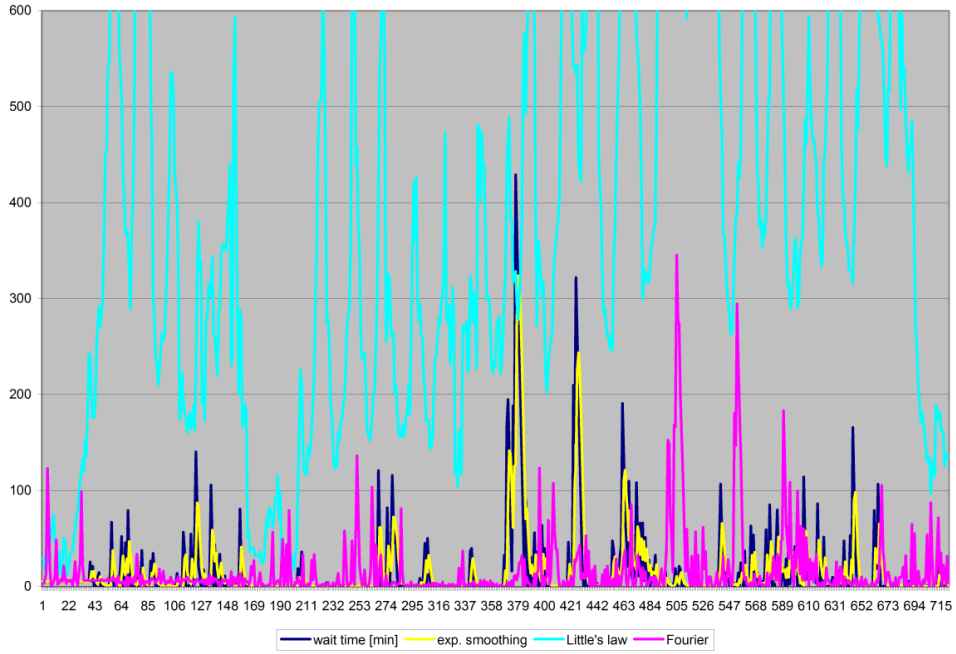

Figure 4.18: Comparison of measured input queue waiting time and estimations for site 5.

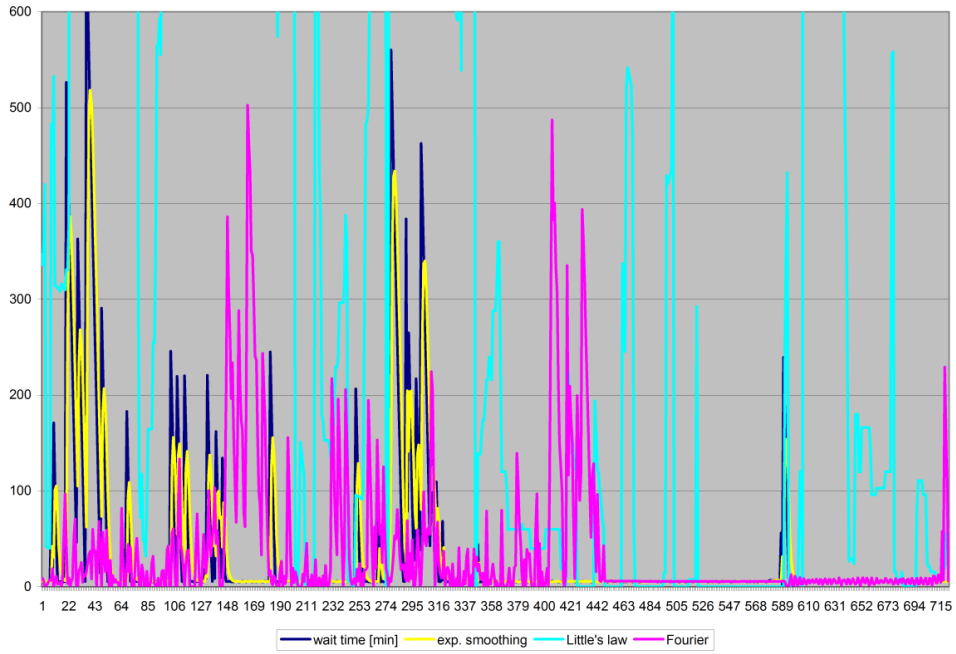

Figure 4.19: Comparison of measured input queue waiting time and estimations for site 6.

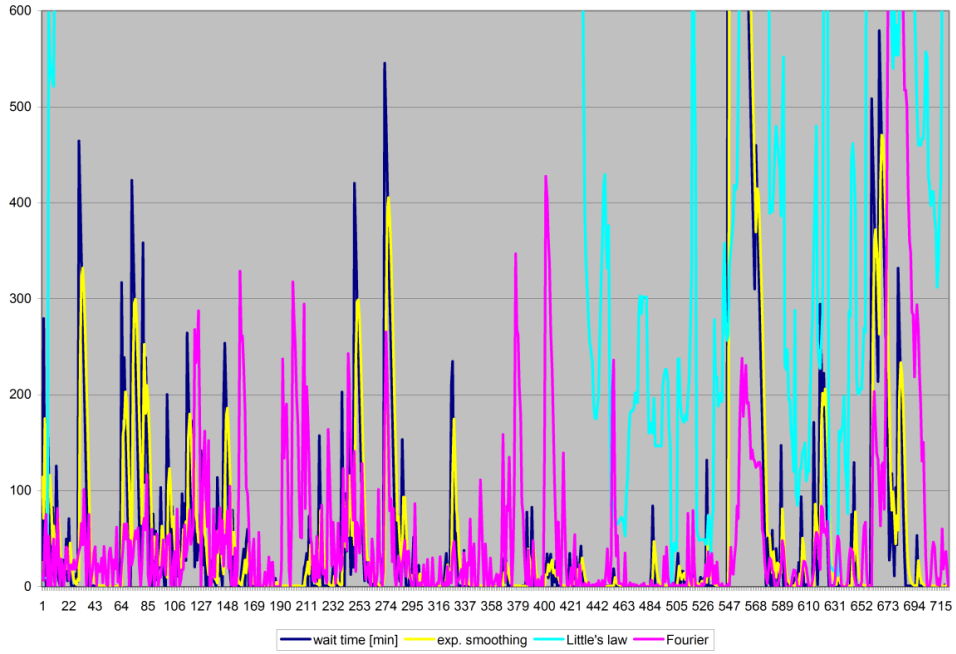

Figure 4.20: Comparison of measured input queue waiting time and estimations for site 7.

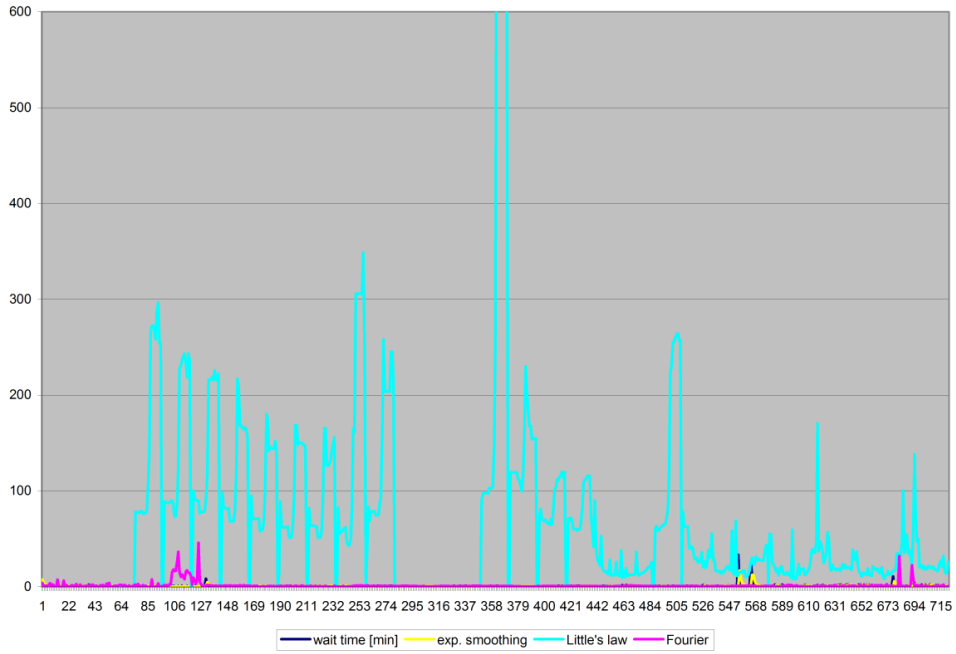

Figure 4.21: Comparison of measured input queue waiting time and estimations for site 8.

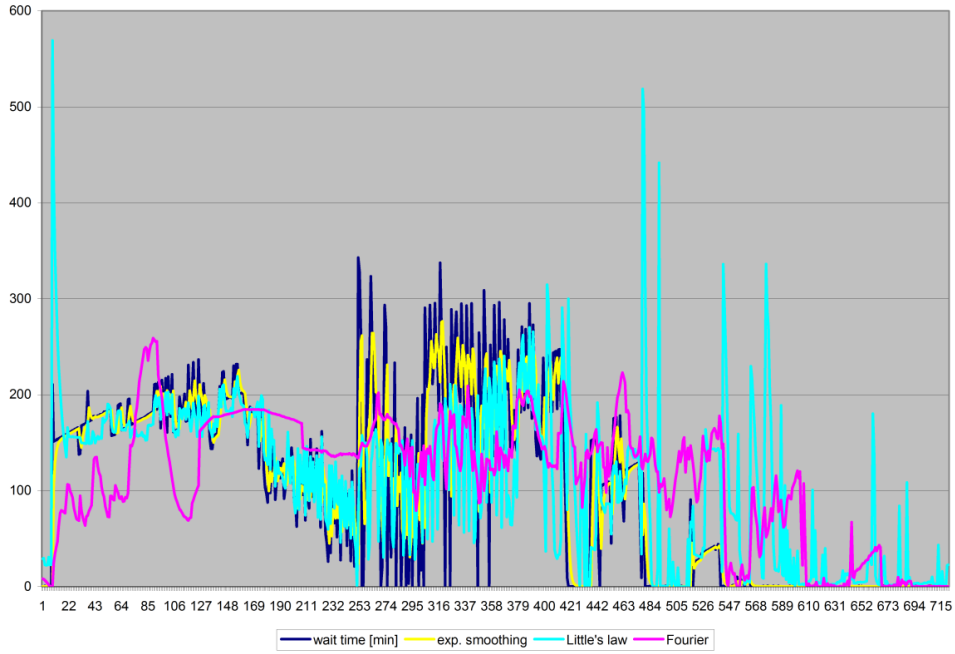

Figure 4.22: Comparison of measured input queue waiting time and estimations for site 9.

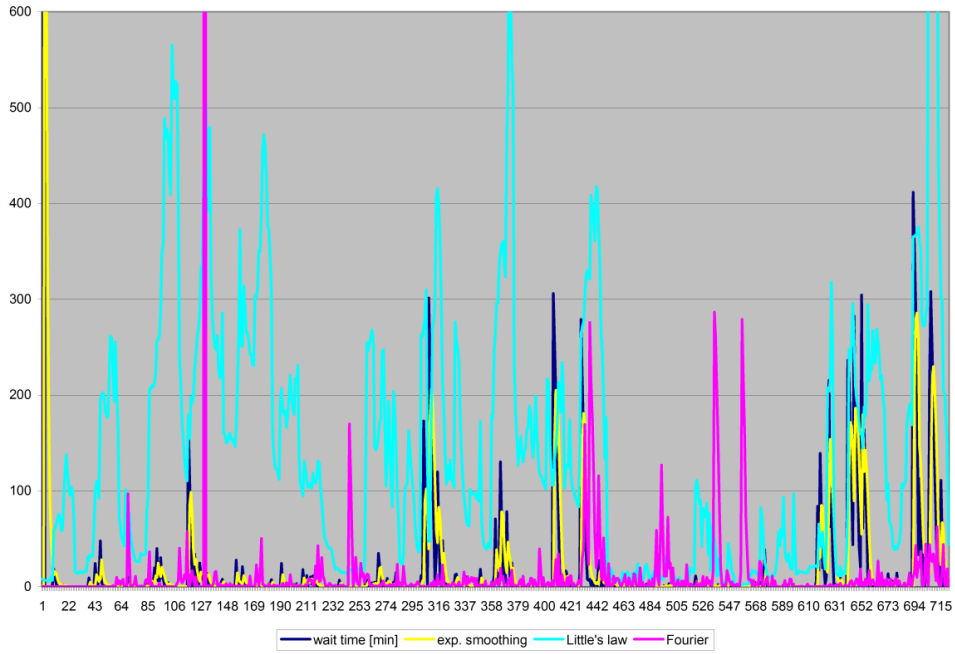

Figure 4.23: Comparison of measured input queue waiting time and estimations for site 10.

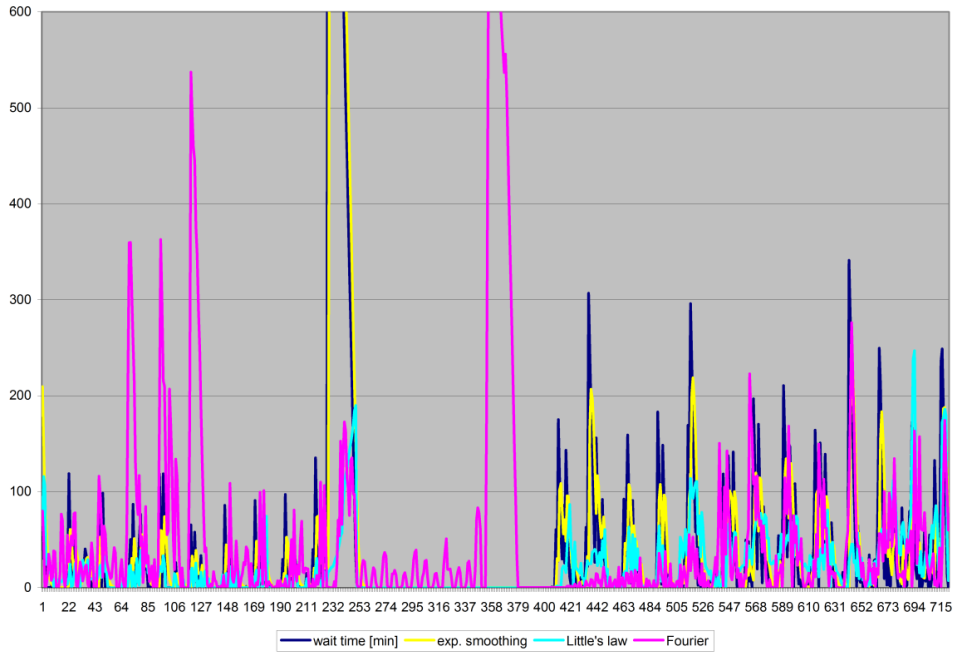

Figure 4.24: Comparison of measured input queue waiting time and estimations for site 11.

useful prediction for site 4. Estimation 3 correctly recognizes the highest and the last peak in the interval but in general it is not applicable to site 4.

For site 5 (see Fig. 4.18), estimation 1 performs reasonable with the same issues as before. Estimation 2 greatly overestimates waiting times throughout the interval and is therefore not applicable to site 5. Estimation 3 provides some correct predictions by chance but fails to detect the peaks in waiting times which occur nonperiodically.

At site 6 (see Fig. 4.19), estimation 1 is the only method that consistently provides reasonable predictions. Estimation 2 performs very bad and is not applicable to site 6. The same applies to estimation 3 which only functions during the long periods with constant waiting times in the last third of the interval.

At site 7 (see Fig. 4.20), the situation is very similar to site 6. Estimation 1 is the only applicable method. Estimation 2 and 3 Estimation 2 do not provide useful predictions for site 7.

For site 8 (see Fig. 4.21), estimation 1 performs very well because the queue waiting times are nearly constant. Peaks are very rare and small so that the

delay in prediction is not a real problem. Estimation 2 gives correct predictions during the periods where its result is zero. In general it is not applicable to site 8. The same applies to estimation 3 which works well for most of the interval due to the almost complete absence of peaks.

For site 9 (see Fig. 4.22), estimation 1 performs reasonable. As with site 2 the smoothing factor is beneficial for the reduction of jitter in the series of waiting times. Estimation 2 performs very well at the beginning of the interval. Afterwards, the magnitude of waiting time is predicted lower than the actual values. At the end of the interval the site's characteristic changes and estimation 2 does not deliver useful predictions anymore. Estimation 3 reproduces the rough trend of waiting time but worsens in the second half of the interval. Overall it is not really applicable as waiting times are not periodic.

At site 10 (see Fig. 4.23), the situation is very similar to site 5. Estimation 1 performs reasonable, but suffers from the inherent delay and deviation of smoothing. Estimation 2 mostly overestimates waiting times and is not applicable. Estimation 3 only provides correct predictions by chance and is not applicable in general to site 10 as peaks occur nonperiodically.

At site 11 (see Fig. 4.24), estimation 1 performs reasonable with its typical limitations. Estimation 2 struggles with the smaller peaks in the first half of the interval but works well in the second half. Even though it is unable to predict the correct height of the peaks, it signalizes them more timely than estimation 1 which is most important for resource selection. Estimation 3 is disturbed by the interruption in the waiting time sequence therefore its performance is best after the 538th hour in the interval. It predicts the following peaks very well regarding both, beginning and height of the peaks. This demonstrates the usefulness of Fourier analysis for sites with periodic patterns in queue waiting times.

The graphical evaluation of the estimation methods shows that all three approaches to estimate input queue waiting times, cover one of the three classes of site characteristics best. Exponential smoothing always performs reasonable and reproduces the trend and the exact value of waiting times best if there are mostly gradual changes in the waiting time series. It is the best choice for resources that employ mainly fairshare scheduling or static priorities. However, exponential smoothing suffers from the delay inherent to first-order filtering and requires frequent measurements of the actual queue waiting times. Queueing theory based

prediction works without delays and is therefore capable to detect changes quickly. It performs the better, the more the LRMS configuration matches the first-come first-served queueing discipline. It becomes unreliable when the arrival rate is very low, or when the actual waiting times are high (more than two hours). Finally, Fourier analysis also avoids the delay of exponential smoothing and is the most suitable method for sites that show strong periodic behavior in queue waiting times. Additionally, it predicts the heights of peaks best if they follow a periodic pattern. Periodicity can occur with fairshare as well as with FCFS scheduling.

During the measurements in Section 4.2 we found that each of the 11 D-Grid sites predominantly matches one of the three classes. Nevertheless, the evaluation of the estimation methods showed that sometimes more than one method can produce good predictions and that estimations can complement one another at the same site. Furthermore, the characteristic of the site can change temporarily or completely as seen at site 9. Therefore, the estimation methods have to be continuously evaluated automatically during run time on their accuracy and applicability to the sites.

4.5 Selection Algorithms

Each of the three estimation methods has its pros and cons which is why we developed additionally three selection algorithms that can dynamically choose the presumably best method for a resource's current situation. All approaches are based on the comparison of predicted input queue waiting times with the actual measured waiting times of the test jobs. To this end, the prediction values of all three estimation methods are continuously calculated at each site. The estimation method with the best recent performance is then selected for the current prediction. There are three methods to determine what the best means.

The first selection algorithm uses the Gauss method of least squares. For each estimation method it calculates the sum of the squared residuals of prediction and measurement over a fixed time interval. The estimation method that has the minimal squared sum in the considered time interval is chosen for the next prediction because the error is the least according to this metric. After the prediction was made, all input values for the Gauss evaluation are updated like in a sliding window. For the time interval we evaluated sizes of one to twelve

measurements. Finally, we confined the interval to the last two measurements, as this gives the best results.

The second selection algorithm incorporates the Gauss method of algorithm 1 and employs additionally a finite-state machine (FSM) with nine states according to Fig. 4.25. It shows a simplified state diagram of the automaton without transition conditions. In each case three states store which of our three estimation methods is currently in use. For each method, an accuracy value is continuously computed by the FSM that comprises the three levels a, b and c, where a is the highest and c is the lowest accuracy. The transitions between the states of the FSM depend on the method of least squares. If the estimation that is currently used has performed best according to Gauss then its accuracy value is increased by one until it reaches the a-grade. Otherwise the accuracy value is decreased as long as it is not lower than c. If the value is already c, the FSM switches to the a-grade of the estimation method that currently performed best. The three-level grading-scale was chosen because each estimation method should be used at least three times in a row to allow for some steady state.

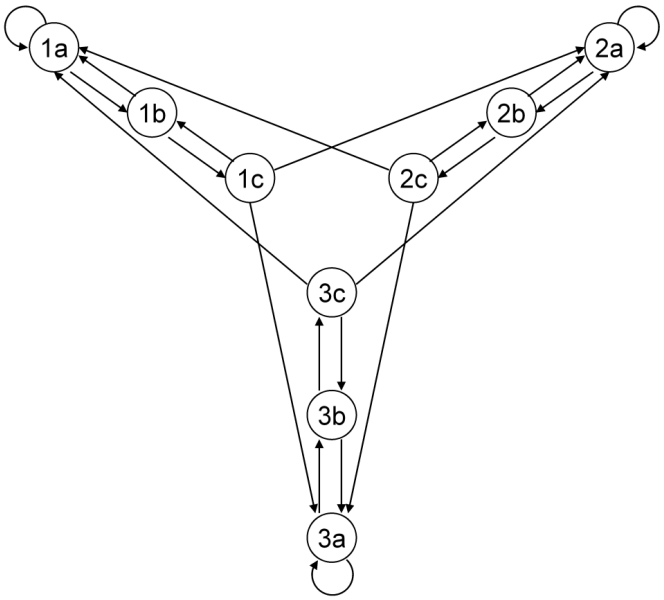

Figure 4.25: Finite-state machine of selection method 2.

The third selection algorithm is based on three coupled FSMs that are run in parallel (Fig. 4.26). Each of the FSMs represents one of the estimation methods. The motivation of this algorithm was to consider each estimation method over

a longer time interval while allowing more frequent method changes at the same time. Therefore, each FSM has 6 states named a to f for the accuracy values (highest to lowest) in order to cover a longer time interval. Fig. 4.26 shows a simplified state diagram of the three automata without transition conditions. Dashed edges depict transitions to other FSMs. To maintain clarity, only the transitions starting from FSM 1 are depicted in Fig. 4.26. FSM 1 is the automaton with states 1a-1f. The others are analog.

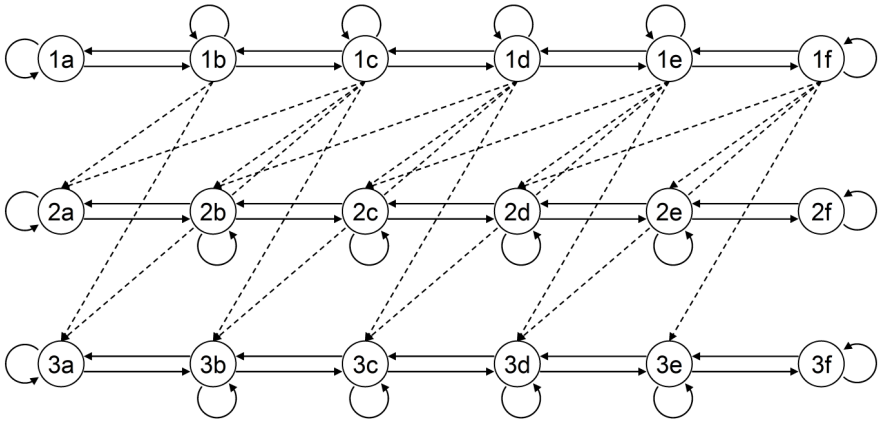

Figure 4.26: Part of the finite-state machine of selection method 3.

Each FSM changes its state depending on the accuracy of its estimation method in the last prediction. The six grades a-f correspond to the following manually chosen deviation between prediction and measurement of queue waiting time, |d|, respectively.

a: $|\mathrm{d}| \leq 1.5$ min,

b: $1.5 \text{ min} < |\mathrm{d}| \leq 4.5$ min,

c: $4.5 \text{ min} < |\mathrm{d}| \leq 13.5$ min,

d: $13.5 \text{ min} < |\mathrm{d}| \leq 40$ min,

e: $40 \text{ min} < |\mathrm{d}| \leq 120$ min,

f: $|\mathrm{d}| > 120$ min.

These deviations have proven most suitable for the 11 D-Grid sites during evaluation (cf. Section 4.6) and yielded the best performance among various setups with 4 to 7 states.

In each computation cycle, the accuracy of each estimation method is updated. If the accuracy of the last prediction was better than the current FSM state indicates, then the grade is increased by one towards grade a, unless it is already a. Otherwise is decreased by one unless it is f. Afterwards, algorithm 3 chooses the method for the next prediction. The current method is kept, unless its accuracy becomes worse than a and one of the two other methods has a better accuracy at the same time. In that case the method with best accuracy is chosen.

In the implementation, the FSMs of the second and third selection algorithm were initially realized with Moore machines [115]. In this kind of finite-state machine the output values are determined solely by its current state. In the selection algorithms the output of the automata is the chosen estimation method. This means that the next prediction is made with the method represented by the current state of the FSM. The state change according to the latest performance of the predictions, i.e. the input of the automata, was processed only afterwards. During our tests this delayed effect turned out to be disadvantageous. Therefore, the implementation of both algorithms was changed to Mealy machines [115]. In this kind of finite-state machine the output values are determined both by its current state and by its current input. For the selection algorithms this means that the chosen estimation method depends on the methods' latest performance. That method is used that is represented by the new state of the FSMs because it is more actual and therefore more accurate.

4.6 Evaluation of Waiting Time Prediction

In the previous Sections we have explained that all estimation methods exponential smoothing, Little's law and Fourier analysis are run in parallel on each cluster and that the most appropriate one is to be chosen dynamically and automatically by a selection algorithm. To make a decision which of the three developed selection algorithms is most appropriate for D-Grid, all three have been evaluated by comparing the selected prediction with measured job waiting times. To this end, the selection algorithms have been integrated into the Java program with

the estimation methods. This means that the program outputs six values for every hour in the provided input range, viz. the three estimation values and the chosen value according to the selection algorithms. The program was applied to the data acquired by the hourly measurements at the D-Grid clusters to perform a retrospective assessment of the final predictions regarding their deviation from the actual queue waiting times.

In the following we present the graphical evaluation of the predictions for all 11 D-Grid sites. We again show the results for the same timeframe of 720 hours (30 days) which was presented in Section 4.2 and Section 4.4. Figs. 4.27-4.37 show the predictions chosen by the Gauss method, the nine-state FSM and the coupled FSMs in comparison to the measured queue waiting times where the y-axis denotes the time in minutes.

The graphs show that all three selection algorithms perform similar and that queue waiting time prediction works very well for most of the sites. Upon close inspection it can be seen that the algorithms select predictions from all three estimation methods where they are appropriate so that the estimations complement each other.

The periods with no or very small queue waiting times are predicted correctly with only few exceptions. Sometimes one or two of the selection algorithms falsely predict peaks even though waiting time is low in the measured sequence. Selection algorithm 2 seems to be more prone to this problem than the others. However, false positives in peak prediction are a minor problem for resource selection as long as many good resources are available.

Almost all peaks in queue waiting times are recognized, the figures show only a few missed peaks across the sites. On the downside, recognition of peaks usually happens with some delay. This particularly applies to the sites where exponential smoothing is the only working estimation method. Furthermore, many peaks are predicted with incorrect height. However, this is a secondary issue as long as the predicted height is sufficient to exclude the site in resource selection.

Overall the pure Gauss method of least squares and algorithm 3 with coupled FSMs show a somewhat better performance than algorithm 2.

Besides the visual analysis of the results, we also performed a numerical, i.e. more formal, evaluation with a larger set of collected measurement data. This

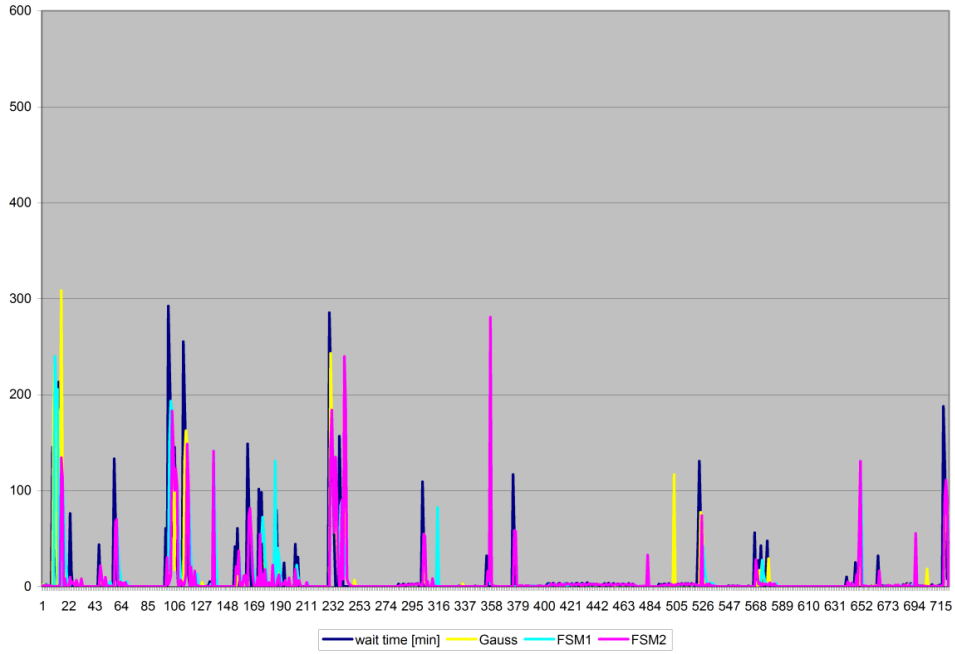

Figure 4.27: Comparison of measured input-queue waiting-time and predictions for site 1.

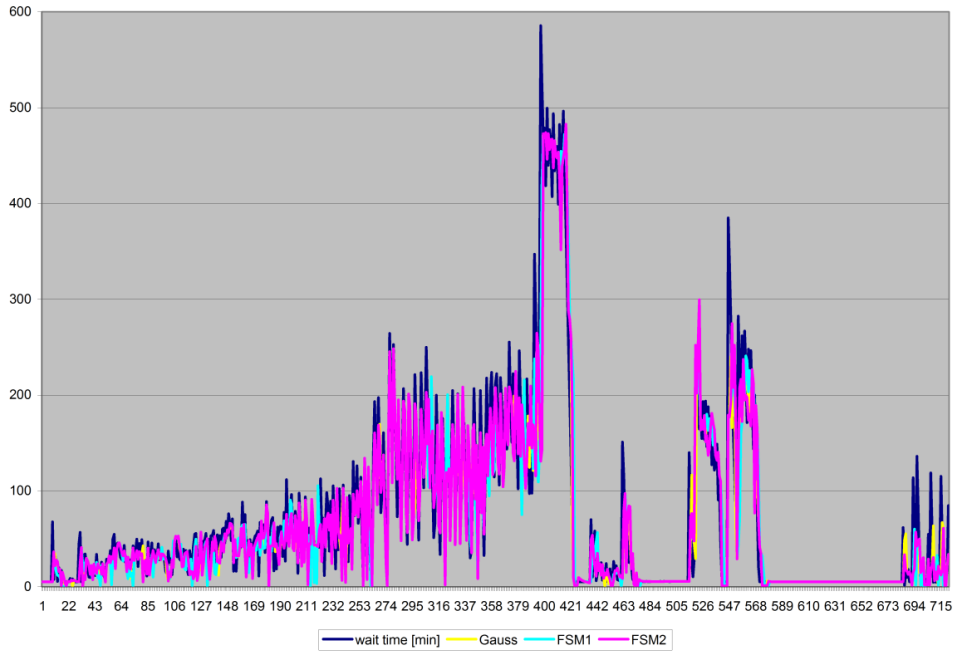

Figure 4.28: Comparison of measured input-queue waiting-time and predictions for site 2.

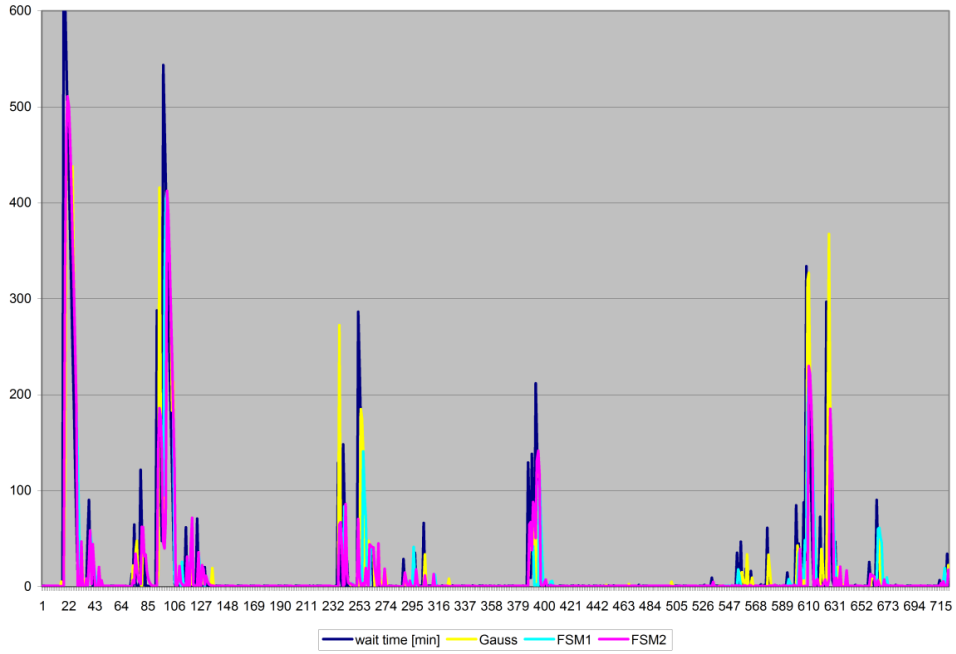

Figure 4.29: Comparison of measured input-queue waiting-time and predictions for site 3.

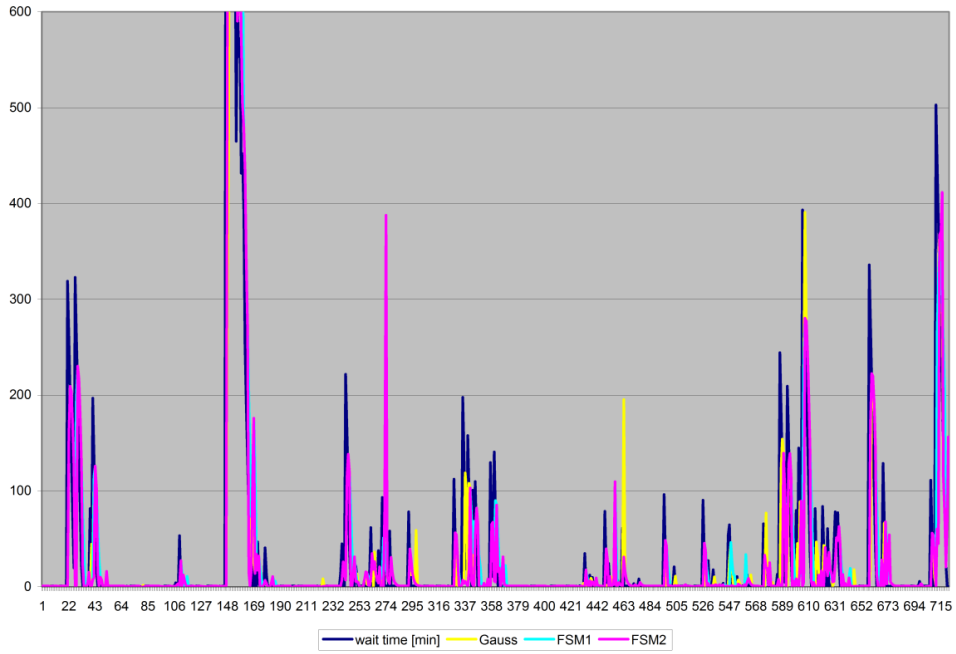

Figure 4.30: Comparison of measured input-queue waiting-time and predictions for site 4.

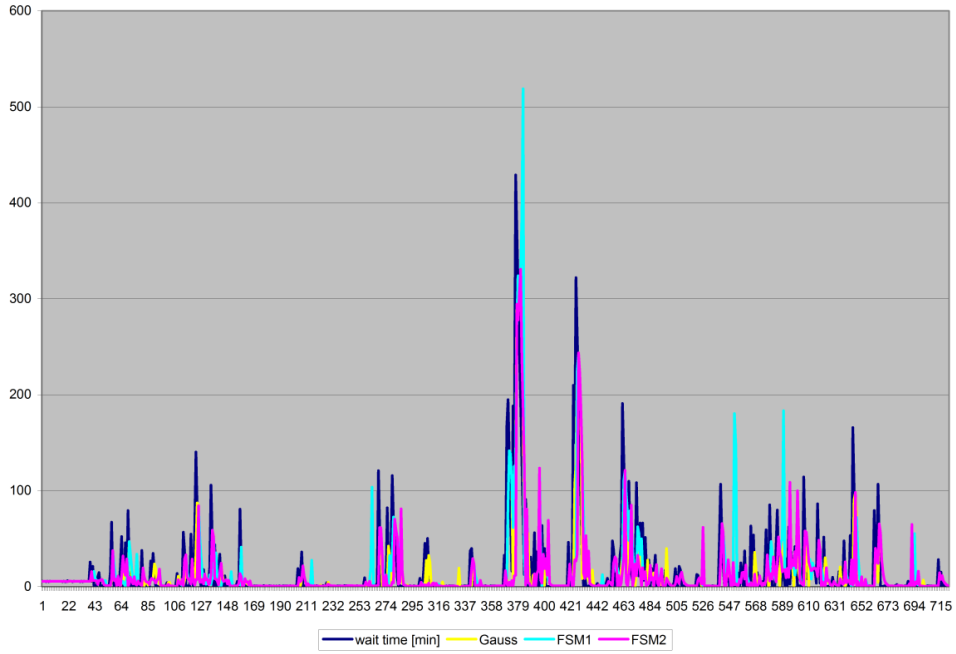

Figure 4.31: Comparison of measured input-queue waiting-time and predictions for site 5.

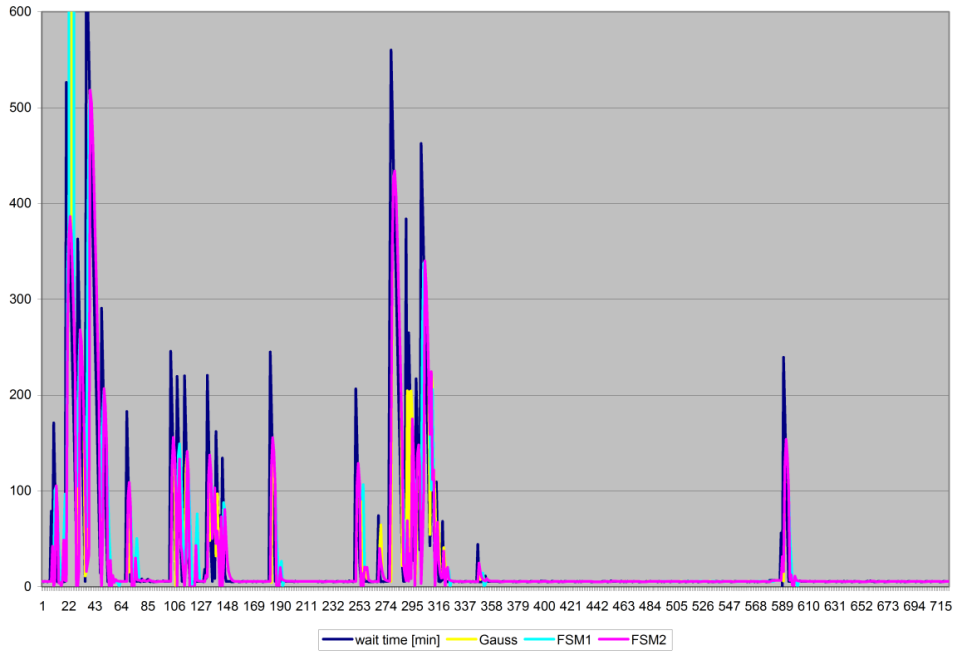

Figure 4.32: Comparison of measured input-queue waiting-time and predictions for site 6.

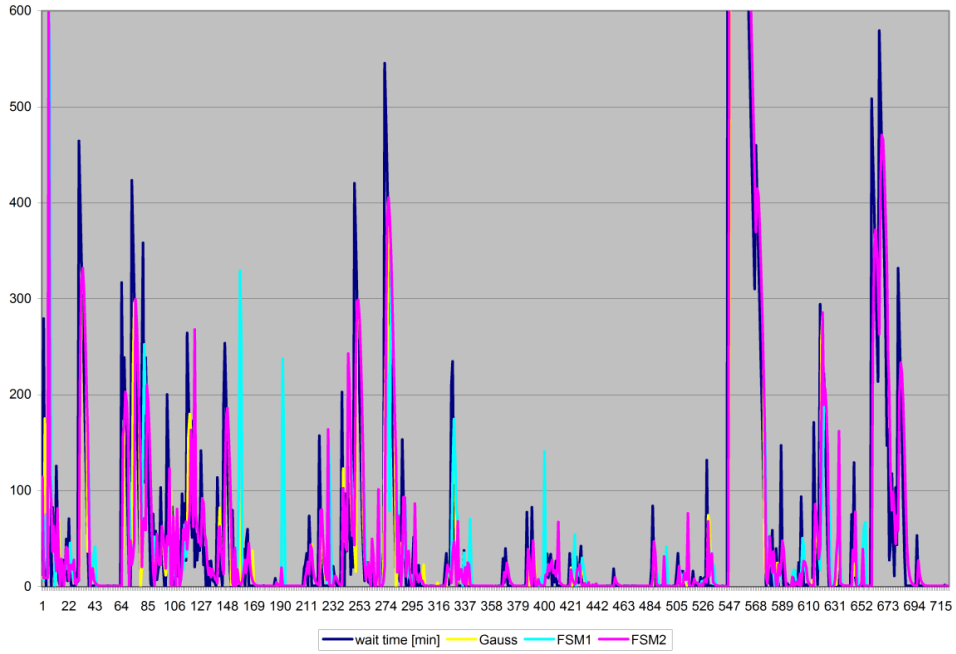

Figure 4.33: Comparison of measured input-queue waiting-time and predictions for site 7.

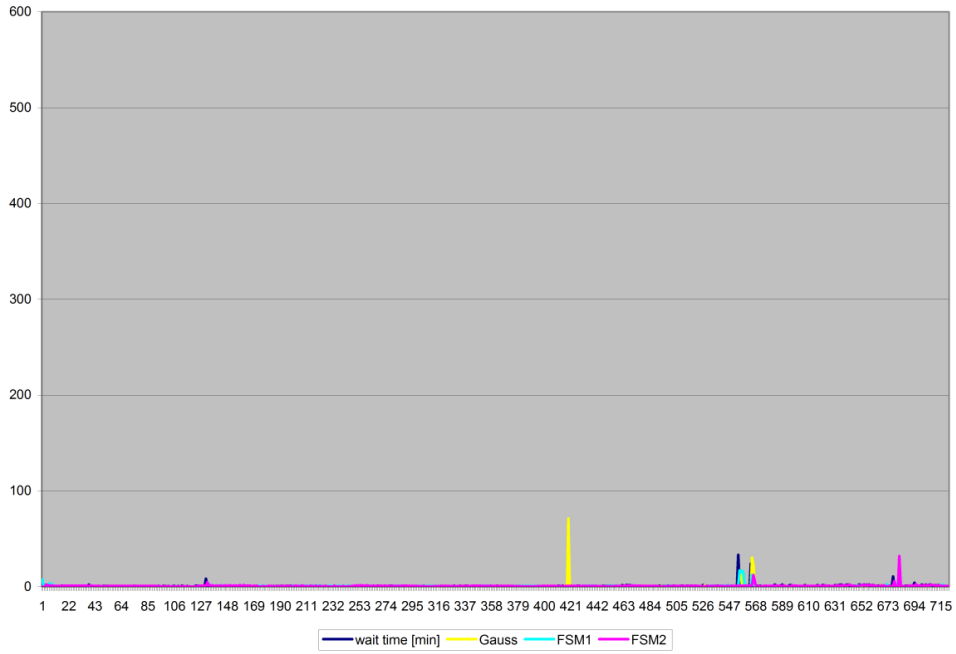

Figure 4.34: Comparison of measured input-queue waiting-time and predictions for site 8.

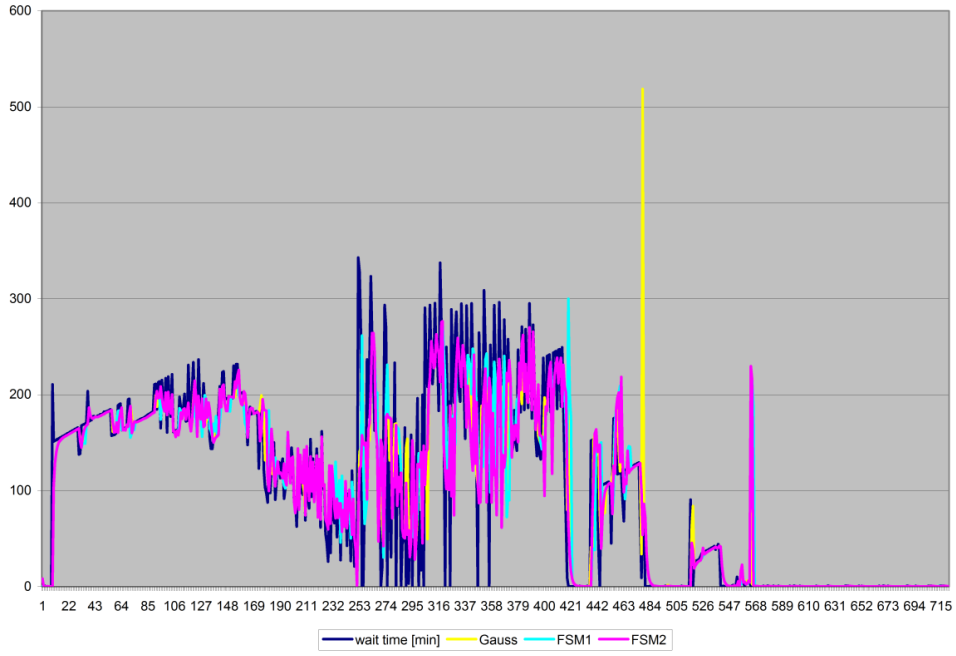

Figure 4.35: Comparison of measured input-queue waiting-time and predictions for site 9.

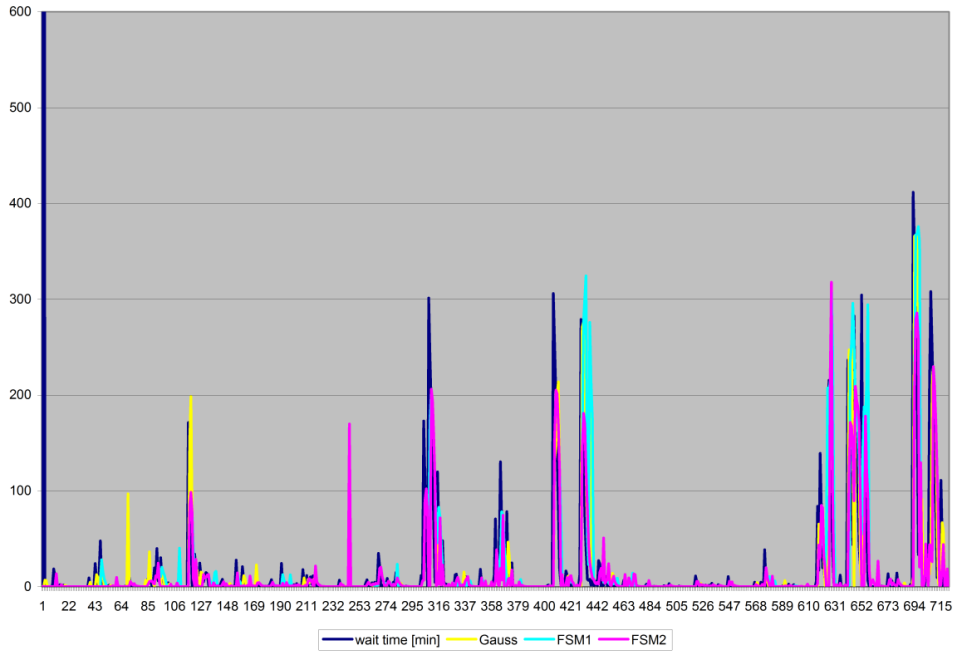

Figure 4.36: Comparison of measured input-queue waiting-time and predictions for site 10.

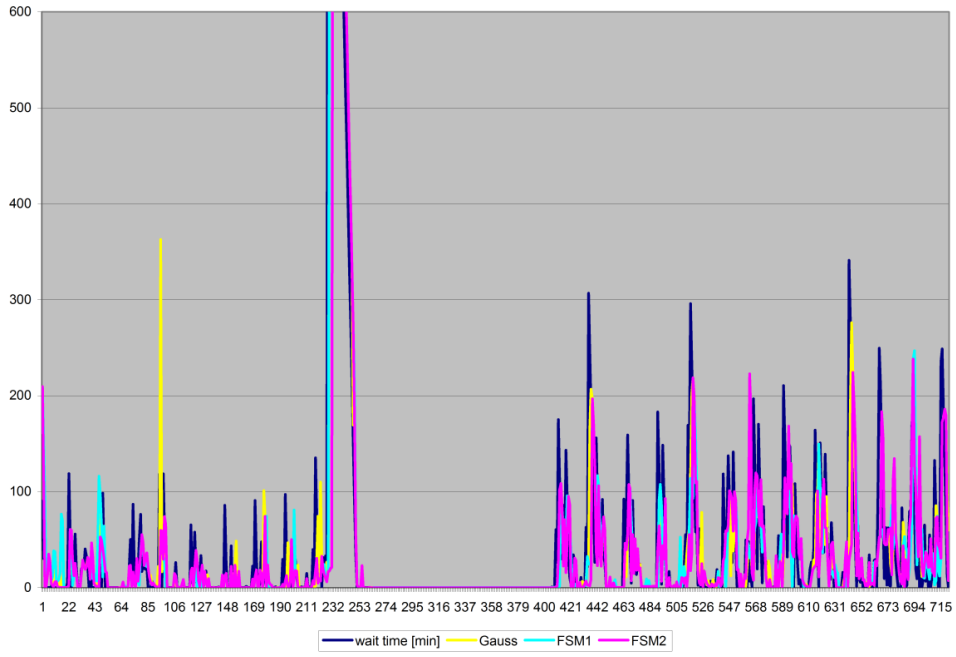

Figure 4.37: Comparison of measured input-queue waiting-time and predictions for site 11.

allowed for an easier and more thorough assessment of each selection algorithm's performance. At the same time, we used this numerical evaluation for the fine tuning of the parameters associated with the estimation methods and selection algorithms. To this end, we defined four levels of accuracy and count what percentage of predictions falls into which category. The four levels correspond to the following deviation between prediction and measurement, |d|, respectively.

1: $|d| \leq 4$ min,

2: 4 min $< |d| \leq 16$ min,

3: 16 min $< |d| \leq 64$ min,

4: $|d| > 64$ min.

Table 4.1 shows the results of the numerical assessment. The evaluation was performed with data from the same timeframe of 8784 hours (366 days) for all 11 sites. The prediction accuracy of selection algorithms 1 and 3 is almost identical while algorithm 2 performs a bit worse which we had already noticed in the

site	selection algorithm	accuracy (%)			
		$\|d\| \leq 4$ min	$4 < \|d\| \leq 16$	$16 < \|d\| \leq 64$	$\|d\| > 64$ min
1	1	78	7	7	5
	2	76	8	8	6
	3	78	7	7	5
2	1	64	9	12	13
	2	62	9	11	16
	3	64	9	12	14
3	1	78	5	7	8
	2	76	6	8	8
	3	78	5	7	8
4	1	77	7	7	7
	2	75	7	9	7
	3	77	7	7	7
5	1	79	9	7	4
	2	77	9	7	4
	3	79	8	7	4
6	1	66	10	10	12
	2	64	10	10	14
	3	66	10	10	12
7	1	65	7	13	12
	2	64	8	13	13
	3	65	7	13	13
8	1	94	1	0	3
	2	93	1	1	3
	3	94	1	0	3
9	1	66	4	9	19
	2	64	4	9	20
	3	65	4	9	19
10	1	93	4	0	0
	2	93	4	0	0
	3	93	4	0	0
11	1	95	2	1	1
	2	94	2	1	1
	3	95	2	0	1

Table 4.1: Accuracy of queue waiting time predictions in 8784 hs.

graphical examination. Upon close inspection of the absolute number of predictions in each category, algorithm 3 turns out to be slightly better than algorithm 1. Therefore, it was chosen as the final approach.

With algorithm 3, on 7 of the 11 sites more than three quarters of the predictions deviate from reality by not more than 4 minutes. At the 4 other sites, about two thirds of the predictions have the same accuracy. Overall, more than 70% of the predictions deviate from the measured waiting times by not more than 16 minutes. Such deviation values are good, especially if one takes into account that input queue waiting times and execution times can last for many hours as we have seen in Section 4.2. This result proves the robustness of our approach, which despite its complexity overall only takes a few seconds to compute.

4.7 Extension of ResourceUpdater

To make queue waiting time prediction available for MediGRID, it had to be integrated into the MediGRID infrastructure (cf. Fig. 2.3). We achieved this by an extension of the ResourceUpdater component. The ResourceUpdater is a Java program that runs as a daemon on each resource, e.g. a cluster, and periodically updates the information about the resource in the eXist resource database. We integrated our implementation of the three estimation methods into a Java class which reads the output file of the measurement program which is also running at the site. For the selection of the current prediction we integrated the third selection algorithm because it performed best in evaluation.

As a further improvement, the ResourceUpdater does not just return the result from the chosen estimation. As it runs continuously, it can employ additional knowledge to enhance the prediction which is beyond the scope of the estimations. For this purpose, the Java class continuously checks the output of the measurement program for the queue waiting time of the test job from the current hour. As soon as it is available, this value is used as input for estimation method 1 instead of the value from the last hour. This way, estimation 1 is updated twice in the measurement interval. This enhancement is especially useful to signalize the end of a peak.

On the other hand, the Java class also notices when the predicted queue waiting time of the test jobs is being exceeded. In that case the actual current waiting

time of the eldest queued job is returned instead of the prediction. For distinction of cases the ResourceUpdater then returns the negative value. This helps to recognize peaks in the waiting time already when they begin.

It should be pointed out that both improvements could not be reflected in the assessment in Table 4.1. This means that in practice queue waiting time prediction performs even better than described in the previous section. The ResourceUpdater stores the result in the resource database in the new property `"State.QueueWaittime"` within the hardware resource description (cf. Fig 2.7) of each cluster.

4.8 Summary Discussion

During our measurements in D-Grid we gained a number of findings and identified four major problems that have to be addressed to enable an efficient meta-scheduling. In the face of this, we developed and implemented a methodology for input-queue waiting-time prediction of clusters that can operate under the boundary conditions of D-Grid.

Computing sites that take part in D-Grid retain their site autonomy. This means that normally no access to internal data of the LRMS, e.g. site scheduler statistics, is available. Additionally, there is no obligation for sites to install supplementary software to supply such information to Grid schedulers. Therefore, the essential characteristic of our prediction methodology is that it only requires regular user permissions. No support from the computing sites is necessary which ensures a wide and flexible applicability. The measurement program as well as the ResourceUpdater can be installed by any Grid user without assistance from the site administrators. Afterwards, both programs can be run under that respective user account. They do not depend on administrative privileges to gain insight into the schedule of the LRMS, which many D-Grid sites would not allow because their resources are not exclusively used for D-Grid. Furthermore, our methodology works with the existing LRMS that are installed at the sites. Only small changes in the measurement program are necessary to adapt to the command syntax of typical LRMS like PBS and LSF [47]. Changes to the LRMS itself are not necessary which is important because these changes would have to be accepted by all sites.

The prediction of input-queue waiting-time is based on three disjoint estimation methods that complement each other. Exponential smoothing represents the class of weighted moving average techniques and is a typical forecasting method for time series. Besides, we additionally employ queueing theory and Fourier analysis for which we did not find any reference in related works. So this is something novel what we did. Furthermore, we evaluated three selection algorithms to automatically select the estimation method for the current prediction. The first algorithm is the classical Gauss method of least squares. The others are two self-developed algorithms that are based on finite-state machines. The evaluation showed that prediction works very well on most of the examined D-Grid sites and recognizes the peaks in waiting times.

With our prediction methodology we are able to attenuate three of the four postulated problems. The lack-of-control problem is addressed by the applicability of the predictions with restricted user permissions. The information-insufficiency problem is remedied by the ResourceUpdater monitoring daemons that provide additional and consistent information for Grid scheduling that is not available from the LRMS itself. Finally, we tackle the non-continuously differentiable-function problem by providing three complementary estimation methods that target different specific site characteristics. The competing-schedulers problem is addressed by our novel MediGRID scheduler that is described in the next Chapter.

Chapter 5

Two-Tier Scheduling Approach

5.1 Previous Scheduling in MediGRID

As we have explained in Chapter 2, the scheduling in MediGRID is accomplished by the scheduler integrated in GWES (cf. Fig. 2.3) that assigns job executions to the resources in the Grid. By the time a Petri net transition is ready to execute, i.e. when all input data are available, GWES automatically selects one of the suitable machines based on current utilization. To this end, machine utilizations are retrieved from the eXist resource database. Then, the scheduling algorithm calculates a quality value, i.e. a metric, between 0 (busy) and 1 (idle) for all resources. The calculation differs for different types of resources. Its purpose is to make resource utilization comparable across different resource types. For workstations, e.g., the calculation is based on the Unix CPU load during the last minute [54]. In the end, the scheduler chooses one resource from the sublist of resources whose metric is greater than a given threshold value (e.g. 0.4). The selection is randomized to prevent an overload of machines especially in case of outdated or insufficient monitoring data. This way, the previous scheduler provides a rough load-balancing between resources and tries to improve the job throughput of the Grid.

The previous scheduler implementation in GWES belongs to the class of just-in-time algorithms, which make the planning only on locally reasonable decisions. Research shows [116] that just-in-time algorithms can produce good results for independent Grid jobs and rather simple workflows, provided that accurate re-

source performance information is available. However, they do not provide any full workflow analysis for task dependencies, i.e. they do not consider the order of task execution for scheduling. Therefore, just-in-time schedules lack performance in complex application workflows that have many concurrent tasks. This affects especially the important class of unbalanced (asymmetric) workflows with parallel threads that differ significantly in expected thread execution times. In this case, preference has to be given to the longer threads to allow all threads to finish within similar time. The execution of such workflows can be improved by employing a full-ahead scheduling as performed by the HEFT algorithm.

5.2 Tier 1: Workflow-level Scheduling

5.2.1 HEFT

Our new approach is based in its first tier on the Heterogeneous Earliest-Finish-Time algorithm [20]. HEFT is an extension of the classical list scheduling algorithm based on directed acyclic graphs for heterogeneous environments. HEFT is a simple and computationally inexpensive algorithm, which schedules a workflow by "backward"-traversing the directed graph from final tasks to first tasks, constructing an ordered list of tasks, and mapping the tasks to resources.

The HEFT algorithm consists of 3 phases [116]:

1. Weighting: it assigns weights to the nodes and edges in the graph;
2. Ranking: it creates a sorted list of tasks, ordered by their priorities;
3. Mapping: it assigns tasks to resources.

In the weighting phase additional text labels called weights are assigned to all nodes and edges in the workflow graph. The nodes represent tasks while directed edges reflect the "depends on" relation. Each node weight corresponds to a predicted execution time of the respective task, while edge weights correspond to predicted data transfer times between resources. HEFT assumes these predictions to be known. In environments with homogeneous resources, the weights directly reflect the predicted times which are resource-independent. For heterogeneous site

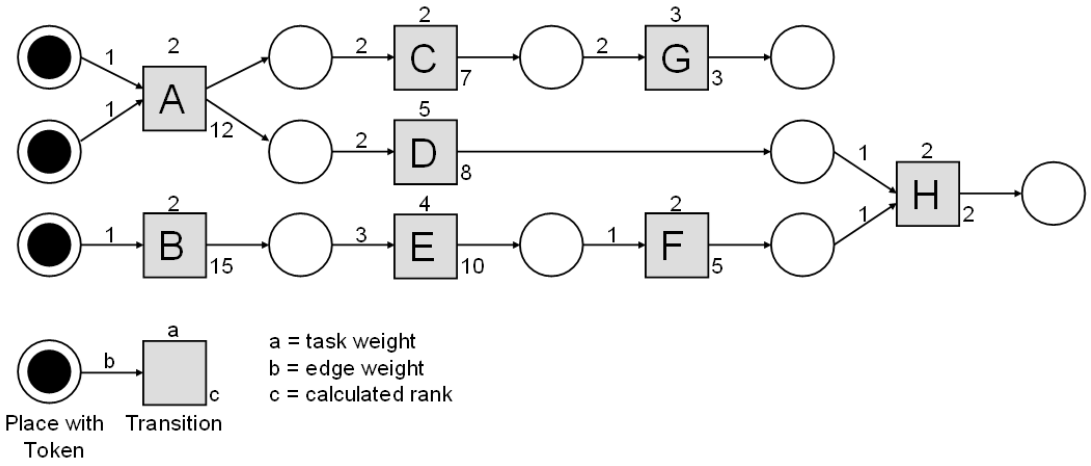

Figure 5.1: Example workflow graph with weights and ranks calculated by HEFT.

resources, the weights must be further processed in order to take into account differences in execution times on different resources, and different data transfer times between sites. Several adjustment methods were proposed and compared [117, 118]. Each of them provides another accuracy with respect to the considered scenario. The common method is to calculate average weights over all resources.

The ranking phase traverses the workflow graph backward from the end nodes to the starting nodes. Therefore, directed graph edges have to be passed in reversed direction. Phase 2 assigns a rank to nodes. A higher rank means greater priority for the task. The rank of a node is equal to the node's own weight plus the maximum successive weight. This means for every edge leaving the node that the edge weight is added to the previously calculated rank of the successive node, and that the highest sum is chosen. In the end, the tasks are sorted by decreasing rank order. This results in an ordered ranking list.

The mapping phase maps tasks from the ranking list to the resources one after the other, such that each task is assigned to that resource which minimizes the task's earliest expected finish time.

Fig. 5.1 shows the ranks of HEFT for a simple example workflow graph. a and b are weights and c is the calculated rank. a denotes the average predicted execution time, and b denotes the average predicted data transfer time. In the example, the weights are arbitrary. The ranking of tasks is: B, A, E, D, C, F, G and H.

5.2.2 Adaptations of HEFT

For the application on Petri net graphs, some modifications of the HEFT algorithm are necessary. Transitions are not connected directly with each other, but are separated by places. Grid data transfer is performed only when a token moves from an input place to the successive transition. That is why the weight of the transfer is written atop the input edges of the transition. Output edges do not have weight values as they stand for program output into the local file system.

In contrast to DAGs of workflows, Petri nets may contain control flow transitions which contain conditions for firing. Furthermore, Petri nets can have loops. Both elements can only be evaluated at runtime. When the full-ahead workflow-scheduling is applied, control flow constructs are ignored, because evaluation cannot be made beforehand. This way, the Petri net branch with the higher weight will be considered for the ranking. If the branch leads to a loop, the loop has to be detected to prevent the traversal of the graph from looping forever. For the ranking, we assume the transitions in the loop to be executed exactly once.

The value of the predicted execution time a of a transition is derived from historical information about previous executions of the same task. For this purpose, the execution time of each task (without queueing delay) is measured during workflow execution. In production use, programs are typically run with similar input data and parameters over a period of time, which results in similar execution times during that time. Nevertheless, execution times can vary from run to run, which is why we smooth the past measured times with a low-pass filter in order to better predict mean values for the future. The filter employs again (cf. Section 4.3) the method of exponential smoothing which implements a low pass filter of first order. It gives an estimation for the next execution times due to previous values. The prediction value is calculated by $a_t = \alpha * m + (1 - \alpha) * a_{t-1}$. Therein m denotes the latest measured execution time, a_{t-1} is the previously predicted value of a, and $\alpha \in [0,1]$ is called smoothing factor. We use $\alpha = 0.3$ as parameter. This factor was found as optimal by conducting multiple field experiments. It emphasizes previously computed values over new ones for a better smoothing but also reproduces long-term changes in execution times. The method of exponential smoothing facilitates the integration into the existing GWES environment because few additional values have to be incorporated into the D-GRDL description of the respective software in the resource database. For each machine the

current value of a and the number of executions are stored in the database. In the weighting phase of HEFT the predicted execution time is then calculated as the weighted average of a over all machines. An alternative procedure would be to normalize the latest measured execution time m with respect to a reference resource by means of a performance rating for each machine.

The data transfer time b depends on the data size and the throughput of the network link connecting the resources. Predicting these values ahead of the workflow execution is impossible because neither data size nor the sites between which the data transfer will take place are known. Additionally, data transfer rates can vary during workflow execution time. Because of this, prediction of b would require a large set of measurements covering multiple previous values for all combinations of sites. This would imply major changes in the resource database. Therefore, we assume instead for the first tier that data transfer time behaves as a normally-distributed random-variable with a mean of 10 seconds and a variance 4. This assumption is also supported by practical experience.

Another aspect that has to be considered are the potential input-queue waiting-times that are imposed on tasks and that lead to a prolongation of pre-execution delay. Many Grid-scheduling research-groups assume a Grid model with high availability and good control over the resources by the Grid scheduler [87], which is often the case for scientific workflows executed in research institutions. However, as our measurements in Section 4.2 have shown, such assumptions do not hold for the D-Grid environment. There, availability of resources is limited and queue waiting times can last up to hours. Additionally, the Grid scheduler has no control over the site-level resource-management systems. These are fundamental constraints that limit the possibilities of meta-scheduling in D-Grid.

These circumstances lead to the conclusion that full-ahead workflow scheduling alone is inappropriate in the D-Grid. Therefore, we only employ the first two phases of the HEFT algorithm to calculate static priorities for the Petri net transitions before the actual processing of the workflow starts. For the mapping phase that assigns tasks to resources, we employ an improved version of the previous dynamic just-in-time scheduling in GWES.

5.3 Tier 2: Grid-level Scheduling

5.3.1 Task Prioritization

The second tier of our approach performs a multi-criteria just-in-time Grid scheduling based on current resource data, performance predictions and additional information available at runtime. During the workflow processing, transitions that are ready to execute, i.e. enabled, are placed in a queue within GWES. While GWES processes many workflows at the same time, the GWES-internal queue holds tasks from several workflows belonging to different users. Initially, the tasks are in the order in which they became ready to execute, i.e. when all preceding transitions were finished.

While the only optimization goal of the workflow-level scheduling is to decrease the execution time of a single workflow, joint Grid-level scheduling of all tasks allows for Grid-wide, i.e. for global improvements. In every scheduling cycle, all tasks in the internal queue are rearranged depending on the prioritization policies and mapped to the available resources. In the previous implementation, the only prioritization option is to assign static priorities to transitions by defining them in the workflow description. In the new approach, the prioritization is split up into hierarchical levels.

For the prototype we realized an implementation of the prioritization with two levels. The lower level is the transition level. On this level, the principle is to sort tasks according to their ranks calculated by the HEFT algorithm. This sorting improves the performance of unbalanced workflows as it optimizes the execution order of tasks from parallel threads within workflows. On the second level we consider the workflows to which the tasks belong to. This workflow level preserves the original order of the workflows that the tasks originate from. For example, if the internal queue contains three tasks, one task t_1 from a workflow w_1, followed by one task t_2 from a workflow w_2, followed by another task t_3 from workflow w_1, this order of workflows will remain the same. Tasks t_1 and t_3 which both originate from w_1 however, may switch their positions depending on their rank. This procedure provides equality between workflows as it prevents workflows that were started later from delaying the earlier ones. That would happen if tasks were sorted regardless of their originating workflows because HEFT assigns the highest ranks to the first tasks within a workflow.

Additionally, we propose an extended, more rigorous procedure that uses four hierarchy levels: transition, workflow, user, and urgency level to allow advanced control of prioritization. An example of this procedure is shown in Fig. 5.2. On the transition level, the default principle is to sort all tasks according to their ranks calculated in phase 1 and 2 of the HEFT algorithm. Alternatively, tasks can be sorted by static priorities that are defined in the workflow description.

After the first sorting, all tasks are sorted a second time with regard to the workflows they originate from. Workflows are sorted chronologically by default. This means that the first entries in the internal queue will be all tasks from the earliest started workflow, followed by the tasks from the second eldest workflow, and so on. Among the same workflow, tasks keep their positions of the first sorting. This method prioritizes older workflows so that when many workflows are started one after the other, the workflows that are started later do not unnecessarily delay the first ones. This would happen, again, if tasks were not sorted by workflows due to the higher ranks that HEFT assigns to the first tasks of workflows. Alternatively, workflows can be sorted also by static priorities that are defined in the workflow description.

The third level sorts the tasks with respect to the users who submitted the workflows. To improve fairness between MediGRID users, a fairshare technique can be employed that continuously accounts resource usage within a fairshare window of e.g. 14 days. Then, tasks from different users are sorted in ascending order with the users' resource utilization. As an alternative, static priorities can be defined for MediGRID users. The order regarding workflows and ranks from the previous levels is again preserved.

The final level treats tasks from time-critical workflows and places them at the beginning of the internal queue. Such tasks can e.g. belong to interactive workflows that are marked with a special "urgent" flag. All four levels can also be disabled which means no sorting will be done for transition, workflow, user or urgency criteria.

The decision about the definite prioritization policy is rather political than technical. The determination of policies is a task of the VO management. Therefore, the prototype employs the simpler first procedure for the time being.

It should be pointed out that in each case the prioritization only affects the transitions of one scheduling cycle. This means, their number depends on the fre-

Attribute	Priority (decreasing →)				
transition	t_4	t_5	t_2	t_1	t_3
workflow	w_1	w_2	w_3		
user	u_2	u_1			
urgency	t_3				

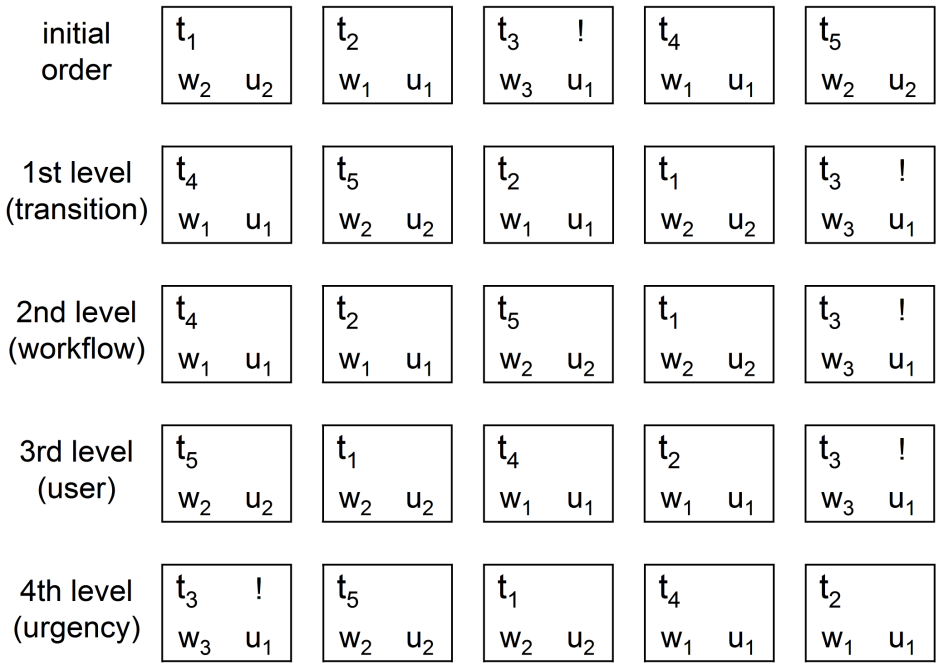

Figure 5.2: Example of four-level transition prioritization procedure with default sorting. The Figure shows the order of the transitions after each level. Transitions will be executed from left to right.

quency of scheduling, the number of concurrent workflows, the parallelism within workflows, and the number and average length of tasks per workflow. The more transitions accumulate in the internal queue, the more important prioritization becomes.

After the prioritization, the tasks are submitted by GWES in their final order to the best resources currently available. Ideally, no tasks are retained in the internal queue of GWES if sufficient resources are present. Our most important goal in resource selection is the reduction of task execution time that is extended by Grid-wide optimizations. Similar to the previous implementation, the scheduler does balance load across the resources and improve thereby job throughput.

5.3.2 Resource Selection

As described in Section 5.1, the previous scheduler in GWES assigns each task to one of the machines whose quality value is greater than the threshold. The selection is randomized to prevent an overloading of the assumedly best machines and as precaution against insufficient utilization information. To still achieve some grading of resources, the scheduler groups machines with roughly similar quality values into so-called clusters. Within each cluster the order of machines is randomized, the quality clusters among themselves are sorted by decreasing order of qualities. The allocation of the tasks in the internal queue starts with the machines of the first cluster, followed by the machines of the second cluster and so on.

For the new approach we performed a more targeted consideration of resource selection. As the main goal of resource selection is the reduction of task execution time, the composition of the turnaround time of Grid jobs must be analyzed. It consists of the actual processing time and pre-execution delay.

Processing time of MediGRID jobs depends on the type of CPU that is integrated in the employed machine. Average execution times of each program are known for each machine where the program was executed in the past. They are already stored in the resource database and used for the prediction of execution times in the weighting phase of tier 1. Likewise these values could be used to determine the fastest resource for each job.

Alternatively, processing time could be optimized by introducing a performance

rating for all compute resources in the Grid. A typical example of such a rating are SPEC values [119]. Even though the SPEC benchmarks only allow for a limited classification of machines, they are suitable in the MediGRID context, as most jobs are neither I/O-intensive nor parallel.

However, on the one hand most D-Grid resources have similar CPU performance. They are equipped with x86-64 processors from the same generation whose per-core performance differs only slightly due to small differences in clock rates. This is because most systems were brought into service at about the same time. On the other hand there is a consensus in MediGRID that scheduling should not prefer any site a priori but utilize all resources. For this reason we do not consider hardware-related performance in the scheduling decision.

The pre-execution delay is made up of several phases. A general factor that adds to pre-execution delay is the middleware overhead that is caused, e.g. by Globus Toolkit 4. It is introduced by middleware components such as the execution management and security infrastructure services involved in job submission. Middleware delay can be neglected in Grid scheduling as this overhead time is almost identical on all sites because all employ the same middleware. Furthermore, job submission overhead is much smaller than processing time of production jobs.

An important aspect in resource selection is the locality of input data. Stage-in of large amounts of input data can take considerable time. Transfer times can be avoided if tasks are scheduled onto compute resources that already have the required data available. However, this is often not possible because computations require several input files that are located on multiple machines so that some data has to be transfered in any case. Another drawback exists if many computations require the same input data that is located on a single machine. If all computations are scheduled onto this machine it might become overloaded. This scenario often exists in parameter sweep application with many parallel threads that process the input data according to differing calculation parameters.

In case the application developer considers it beneficial to execute successive tasks on the same machine to avoid data transfer this can be specified in the workflow description. For this purpose GWorkflowDL allows to define so-called allocation groups. All transitions which are connected by arcs and that belong to the same allocation group are then scheduled on the same resource. This feature of GWES is denoted as spatial “co-scheduling”.

When data transfers are performed to stage-in input files of a task, the transfer delay is determined by the longest of the single transfer times. To minimize this duration, resource selection should prefer such machines that have a fast network connection to the data source. To this end, the data transfer rates between the D-Grid sites have to be measured continuously, e.g. using the Grid benchmarking service Jawari [120]. Our test measurements in D-Grid showed that variance in network throughput is acceptable for short-time predictions. Additionally, we found that the network throughput was usually not equal in both transfer directions between the sites.

During just-in-time scheduling, source and potential destination of the transfer are given. Therefore, it is possible to predict transfer time as $t_{transfer} = t_{setup} + size/throughput_{(source,dest)}$ if the data size is known. However, GT4 does not provide functionality to query the size of files in a file system via a web service invocation. This means that file sizes can only be determined by submitting another job in advance that executes, e.g the Unix "ls -l" command, and transferring its output to the scheduler. Of course this does not only imply a file transfer itself, but it is prohibitive because of the potential waiting time the additional job might experience in the input queue. For this reason we do not concentrate on prediction of transfer delay. For MediGRID this is not very detrimental to scheduling performance though. Large flat file databases are typically deployed in advance by the application developers so that they are available at all sites. Therefore, these transfers are not part of the application workflows.

The dominant pre-execution delay of MediGRID jobs is input-queue waiting-time. As our measurements on D-Grid sites have shown, waiting times can last up to hours. To reduce the overall completion time of tasks which is the sum of pre-execution delay and processing time, resource selection must hence minimize input-queue waiting-time. To this end, we extend the previous algorithm that calculates the quality metrics for all machines. In the new approach, we incorporate the predicted queue waiting times which are determined as described in Chapter 4. Resource selection for each task is performed based on this improved quality metrics. The grading of resources with similar quality into clusters is maintained but the number of quality clusters is increased. This reduces the effect of randomization which is justified by the higher significance of the improved metrics. The allocation of the tasks in the internal queue is carried out as by the

previous scheduler of GWES. Tasks are assigned in decreasing priority order to the machines in the quality clusters which are ordered by decreasing quality.

5.3.3 Quality Metrics for Clusters

Just-in-time Grid scheduling in GWES is based on a quality metric. The calculation formulas differ for different types of resources, e.g. web services, workstations and compute clusters. All formulas return values in [0,1] where 0 denotes that the resource is "busy" and 1 denotes that the resource is "idle". The metrics allow for comparison of the utilization of all types of resources.

For clusters which are managed by a LRMS, the previous function is based on the number of running and waiting jobs,

$$q = \frac{1}{e^{(jobs_{waiting}/jobs_{running})}}.$$

By definition the quality is set to $q = 0$ in case $jobs_{running}$ is 0 to prevent division by zero. The idea behind that formula is that a resource with a high computing power always has many running jobs. The absolute computing power of a cluster is translated by the normalization on the number of running jobs into a relative computing power that reflects the potential availability of that cluster. This makes the site's availability comparable to other sites. The formula returns a quality of $q = 1$ if no job are waiting, and it results in a quality q that approaches 0 if the number of waiting jobs approaches infinity. Likewise, the quality q approaches 0 if the number of running jobs approaches 0. A graph of q is shown in Fig. 5.3.

In practice the formula means that if the quality threshold is set to 0.4, and if 100 jobs are running on the cluster, the scheduler would submit further jobs until 92 jobs are waiting in the queue. Then, the quality drops below the threshold of 0.4 and no further jobs are submitted.

In order to translate waiting times into quality values, we implemented a second formula for the quality metric based on a set of definitions. For an input of 1 minute the quality q' should not be less than 0.9, for 3 minutes approximately 0.8, for 30 minutes approximately 0.4, and for 60 minutes approximately 0.3. If

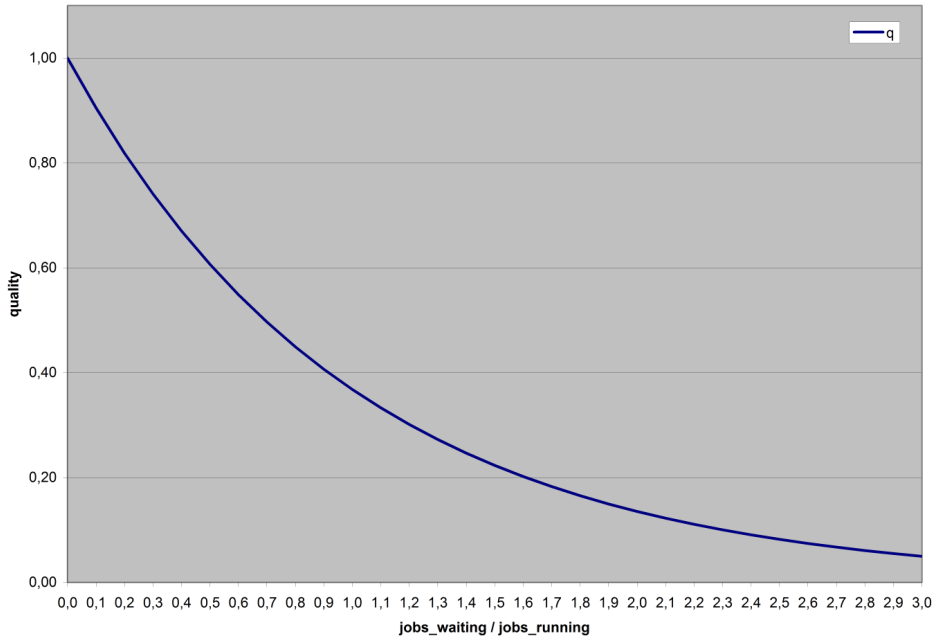

Figure 5.3: Graph of previous quality formula for clusters q.

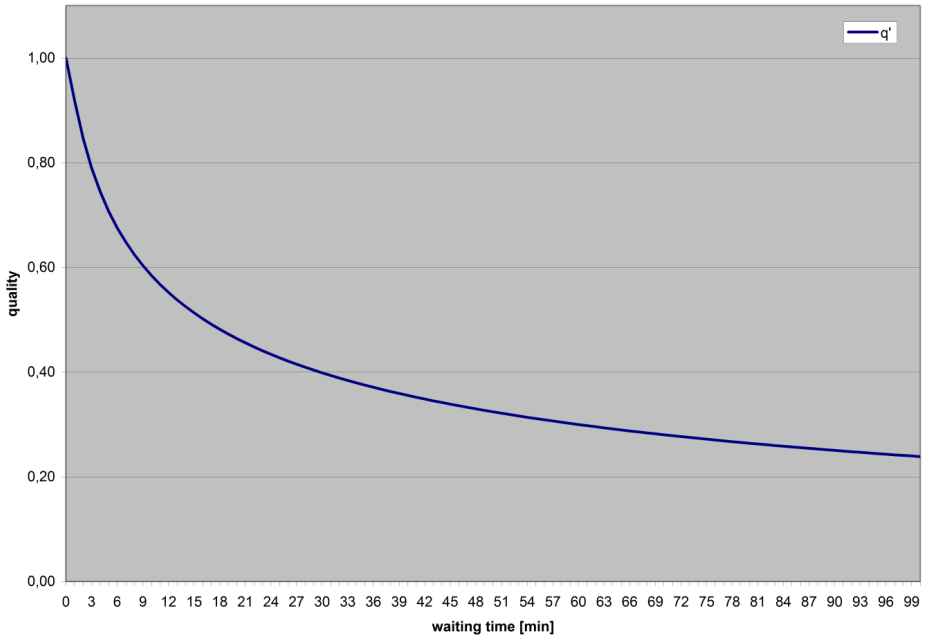

Figure 5.4: Graph of new quality formula for clusters q'.

the waiting time approaches infinity the quality q' should converge to 0. The new function is

$$q' = p^{(\log_2(waittime+1))^r},$$

where the parameters p=0.92 and r=1.5 are defined by means of the definitions. A graph of q' is shown in Fig. 5.4.

In practice this new formula means that if the quality threshold is set to 0.4, the scheduler would submit jobs to clusters with a predicted queue waiting time of up to 29 minutes.

5.4 Implementation in GWES

For the implementation of the two-tier approach we used the then latest GWES release 2.1rc9 as starting point. The GWES source code can be obtained at the official web site [121]. It consists of 166 Java files with more than 700 classes and is organized as an Apache Maven [122] project which simplifies the import into a programming environment and the build of the release.

The implementation started with tier 2, the extension of the Grid-level scheduling. At first the use of queue waiting time predictions was integrated into GWES. The predicted waiting times of each cluster as well as all other properties are retrieved by GWES from the eXist resource database via the database connection class `XMLDB`. The quality metric of each resource is calculated in a class of the scheduler package named `ResourceQualityCalculator`. We extended this class with the new quality formula q'. For all clusters for which waiting time predictions are available in the resource database the new formula is employed. For the rest of the clusters the previous formula q is used. The quality calculation for workstations was not changed.

The previous just-in-time scheduling of GWES is implemented in the class named `RecurrentProrater`. This class conforms to the singleton programming pattern [123] and implements a vector which constitutes the internal queue of GWES. The vector holds all enabled Petri net transitions that have to be mapped onto the available resources. In every invocation of the scheduler, first the resource

alternatives are determined for each transition. Then, one transition is scheduled onto each of the resources respectively. As described in Section 5.3.2, the transitions in the vector are sorted by decreasing priority, and the resources are ordered in quality clusters of decreasing quality.

For the new Grid-level scheduling we improved the RecurrentProrater and implemented a new class named `TwoTierProrater`. In the first step we reviewed the previous code, fixed problems regarding concurrency, improved the robustness, and made several optimizations to reduce processing time, e.g. by avoiding unnecessary operations. In the second step the new features were added. The transitions in the vector now keep a reference to their parent workflows. This allows for the prioritization on workflow level to provide equality between workflows. Additionally, we improved the clustering of resources to reduce the effect of randomization.

The realization of tier 1 started with the implementation of transition execution time prediction. During workflow processing, GWES measures the actual processing time of each transition as well as the overall completion time that includes queue waiting time. The information is aggregated and stored in the eXist resource database within the description of the software resource that the transition refers to. The software resource description contains separate entries for each machine where a program instance was executed. The entries comprise the average program execution time and the number of previous program runs on the specific machine. These statistics are maintained by the workflowanalyzer package of GWES. For the prediction of transition execution times we introduced exponential smoothing in the software resource statistics. It is applied to the measured processing times. The smoothing factor was implemented as $\alpha = 0.3$ as described in Section 5.2.2. The result is stored as an additional value in each of the machine entries.

The workflow-level scheduling was implemented in GWES as preceding step of the Grid-level scheduling. It is performed subsequent to the resource matching as it depends on information about resource alternatives for the transitions. We implemented a new class named `WeightAndRank` as part of the scheduler package which conducts phase 1 and 2 of the HEFT algorithm. The weighting phase iterates over all transitions and retrieves the statistics for each of the machine alternatives from the resource database. The weight of a transition is calculated

as weighted average over all machines:

$$w = \frac{\sum_{m \in machines} t_m \cdot n_m}{\sum_{m \in machines} n_m},$$

where t_m denotes the smoothed value of previous execution times on machine m, and n_m denotes the number of previous executions on machine m. The weights of the arcs are determined as described in Section 5.2.2. After all weights have been calculated, the ranking phase is performed to determine the priorities of all transitions. To this end we have implemented a non-recursive depth-first search algorithm that traverses the workflow graph in reverse direction starting at the final transitions. A hash table is used to keep track of the current traversal path to prevent traversing the graph in cycle. The rank of a transition is the sum of the transition's weight and the weights of all transitions and arcs on the current traversal path. If the transition is reached by several paths, the highest sum becomes the transition's rank. In the end, the ranks are set as transition properties in the workflow description. These properties are evaluated in Grid-level scheduling of the transitions.

Along with the functional improvement of GWES, we also extended the test code for unit testing. The unit tests are run automatically when the GWES release is built by Apache Maven and ensure basic functionality. Additionally, a multitude of logging outputs were added in the code to facilitate thorough testing of the new scheduler.

5.5 Results

5.5.1 Reference Workflow

After the implementation of the two-tier approach we assessed its performance by means of a reference workflow. We opted for a synthetic workflow that combines typical characteristics of MediGRID applications. Therefore, the reference workflow reproduces sequential and parallel task execution in Petri nets. It gives information about the effectiveness of the two-tier approach, even though a synthetic workflow rather allows for a qualitative than a quantitative examination.

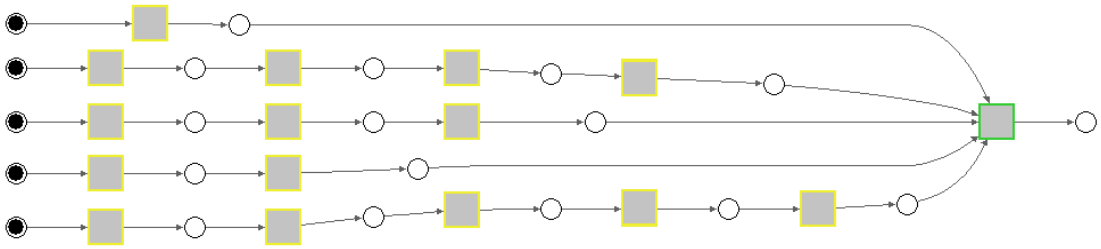

Figure 5.5: Petri net of reference workflow displayed in GWES user interface.

The reference workflow is depicted in Fig. 5.5. It consists of 16 transitions which are divided into 5 threads of 1 to 5 tasks each that all end in a final joining task. Each of the 15 threaded transitions lasts 60 seconds and is scheduled on the clusters which are in production use by MediGRID. The final join transition lasts 10 seconds and is bound to one host to collect the results. The transition execution times had to be short in order not to disturb productive jobs. Nevertheless, even with such short jobs the tests consumed 130 CPU hours on one MediGRID site alone. Short execution times of the jobs emphasize the effect of scheduling on the overall workflow execution time. However, we intentionally did not opt for empty jobs, because GWES determines a job's queue waiting time and execution time via polling, which does not provide a completely precise measurement. If the job lifetime is too short, no state change, e.g. from waiting to running, might be recognized anymore.

5.5.2 Testbed

For the assessment, we configured a testbed similar to the GWES production environment on the MediGRID portal server. The testbed is hosted in a virtual machine with 2 processor cores and 4 GB RAM. The virtual machine conforms to a typical Grid head node with Globus Toolkit 4. The testbed runs three GWES instances in parallel each of which is deployed as a web application in its own Apache Tomcat servlet container. The instances are

1. the original GWES 2.1rc9 (in the following denoted as "RC"),
2. an extended GWES with the new Grid-level scheduling of our approach (denoted as "TT1"),

3. and a further extended GWES which uses Grid- and workflow-level scheduling (denoted as "TT2").

The three separate Tomcat containers became necessary to prevent Java class loading issues. Additionally, the testbed hosts an eXist database which is deployed in another Tomcat container.

During the assessment, all three GWES instances processed the reference workflow simultaneously and competed with each other like Grid schedulers from different user communities do. To provide fair conditions, all instances used the same configuration properties as in production environment. This includes especially the scheduler invocation cycle of 5 seconds (so-called "interval" property), and the minimum waiting time between two job submissions to the same resource which was set to 12 seconds ("wait" property). The resource quality threshold was set 0.4 to obtain meaningful results ("min_quality" property). Furthermore, all GWES instances used information from the same resource database for their scheduling decisions. Due to this, the HEFT priorities in workflow-level scheduling had to be calculated and set ahead of workflow execution to obviate problems resulting from the shared access to the database.

5.5.3 Initial Results

In the first test series, we compared all three GWES versions RC, TT1, and TT2 with each other in 106 runs of the reference workflow. The tests were executed over the course of 10 days. In each run, the workflow was started in the GWES instances at almost the same point in time. A small delay of 4 seconds was introduced between the workflow starts to prevent the Globus WS-GRAM in the testbed from overload by simultaneous job submissions. The delay represents a slight advantage for the RC instance because its workflows were always started first. At most two test runs were started per hour with a time distance of 30 minutes. The job timeout for the submitted Grid jobs was set to 16 minutes to restrain interferences between test runs and to prevent overly long waiting times. The first run in every hour in which all three workflows finished successfully was used only for the assessment.

Initially, a huge number of unsuccessful workflows happened due to memory problems with the eXist database, bugs in RC, failures of the Grid resources, or job

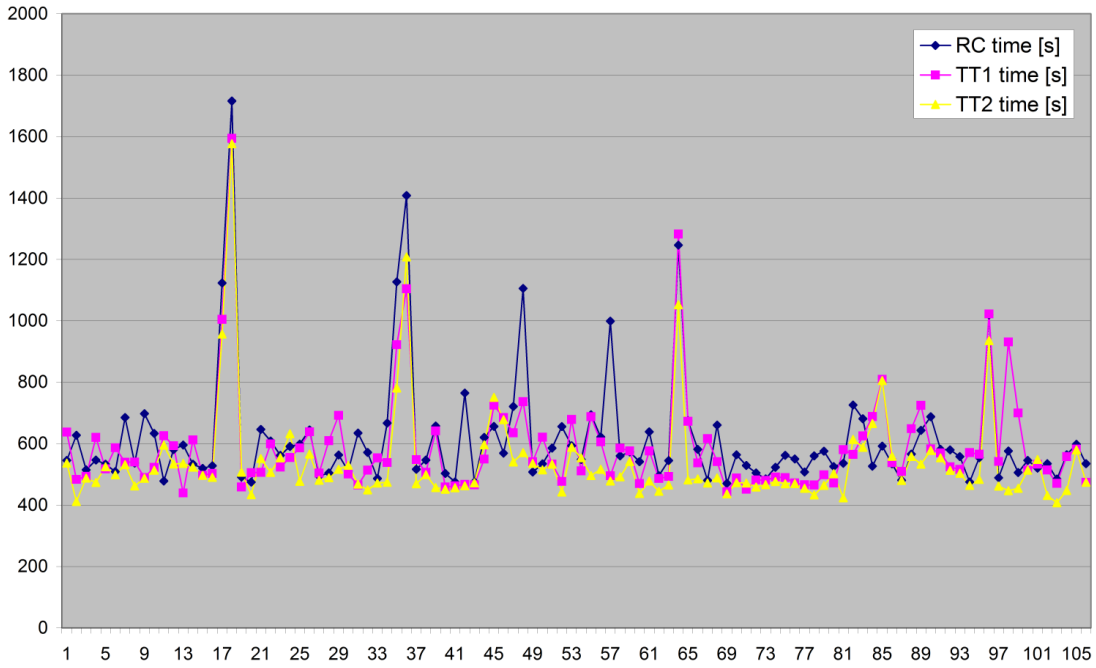

Figure 5.6: Workflow execution times in the first test series.

	RC	TT1	TT2
sum of execution times [s]	65282	62498	57226
arithmetic average [s]	615,87	589,61	539,87
standard deviation	194,55	170,91	160,50

Table 5.1: Evaluation of execution times in series 1.

timeouts. Such unsuccessful workflows were not considered for the evaluation. One reason for this is that sometimes unsuccessful workflows had shorter execution times than successful workflows from the same test.

Fig. 5.6 shows the workflow execution times of all instances in the first test series. The numerical assessment is given in table 5.1. Overall TT1 has shown a speedup over RC of 1.04, TT2 has a speedup over RC of 1.14. The speedup of TT2 over TT1 is 1.09.

Even though the qualitative improvement is visible in the graphs, the speedup is small. The reason for this result is that most workflows were executed without much waiting time in the input queues because the MediGRID resources were not highly utilized during the first test series due to the absence of a major MediGRID

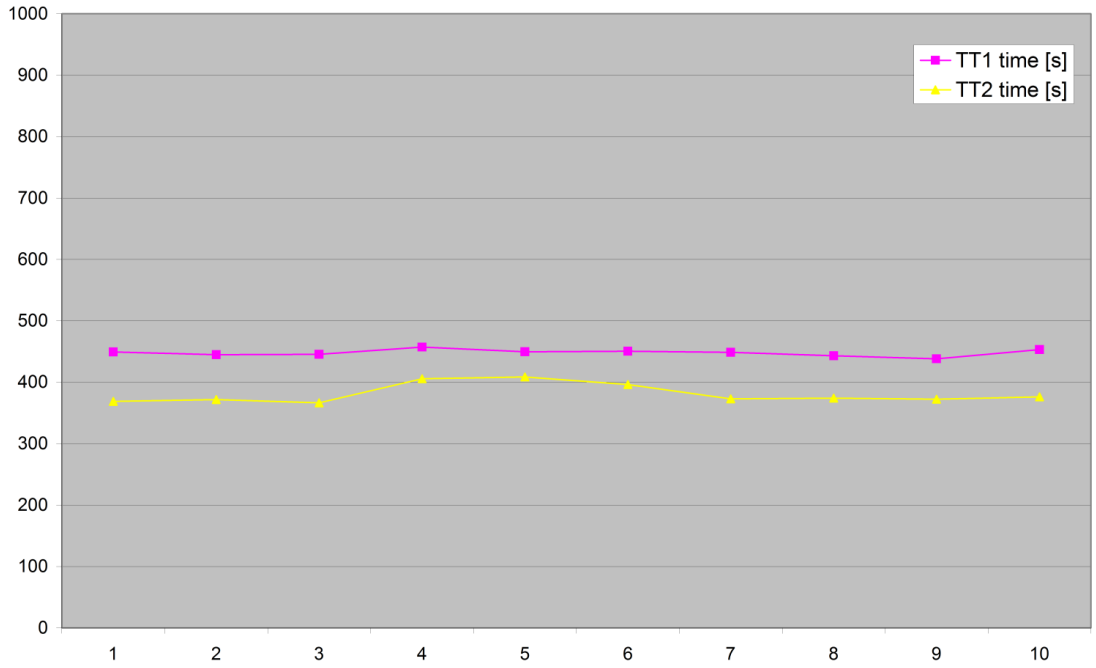

Figure 5.7: Workflow execution times in the second test series.

use case. Under such circumstances also the previous quality formula q performed very well. Additionally, it detected changes quickly because of its high update frequency. In contrast, the waiting time prediction made some false predictions because of its long prediction interval.

5.5.4 Advanced Results

In the second test series, we compared the GWES versions TT1 and TT2 in 10 test runs. The aim of this test was to investigate the effect of workflow-level scheduling. For series 2, only the testbed system was used which does not have queue waiting time because transitions are executed as fork jobs. The reference workflow was started by both GWES instances but in alternating time slots, and only after the other instance had finished workflow processing. This means that only one workflow was running at a time and there was little variance in execution times.

The workflow executions times are shown in Fig. 5.7, the numerical results are given in table 5.2. The benefit of tier 1 is clearly visible, overall it delivered a

	TT1	TT2
sum of execution times [s]	4480	3813
arithmetic average [s]	448,01	381,31
standard deviation	5,41	15,86

Table 5.2: Evaluation of execution times in series 2.

speedup of 1.17 for the given scenario. This speedup is the maximum acceleration for the reference workflow with the given GWES properties on resources with equal waiting time. If there are more resources or more waiting time, acceleration will be lower because the influence of the scheduler's wait property decreases. In case of resources with different waiting times and correct prediction, acceleration increases because the tasks from the longer threads are executed on the better resources.

5.5.5 Final Results

An in-depth analysis of test series 1 showed that none of the two formulas for the quality of a cluster were able to react on all peaks in the queue waiting times, i.e. either q or q' remained high although there was a significant waiting time. However, most of the peaks were detected either by q or alternatively by q', and the combination of both showed a better hit ratio. Combining both quality metrics can easily be accomplished by multiplying q with q'. The resulting $q''=q \cdot q'$ is again in the interval [0,1] and represents a logical "and" of both ratings. q'' is high only if neither q nor q' detects a peak, and as a consequence, only such resources are selected that have good quality in both quality metrics at the same time. In return a decrease of the threshold value might become necessary.

In the third test series, we compared GWES version RC with a variant of TT1 that uses combined quality q'' in 50 test runs. The scenario was similar to test series 1. The reference workflow was always started in RC and TT1 with 4 seconds delay. The test runs were performed with a time distance of 30 minutes, and the job timeout for the submitted Grid jobs was again set to 16 minutes to restrain interferences between test runs. But in contrast to series 1, workflows were also considered for assessment when they were terminated by transition timeouts to include the timeout problem. In case of other errors workflows were

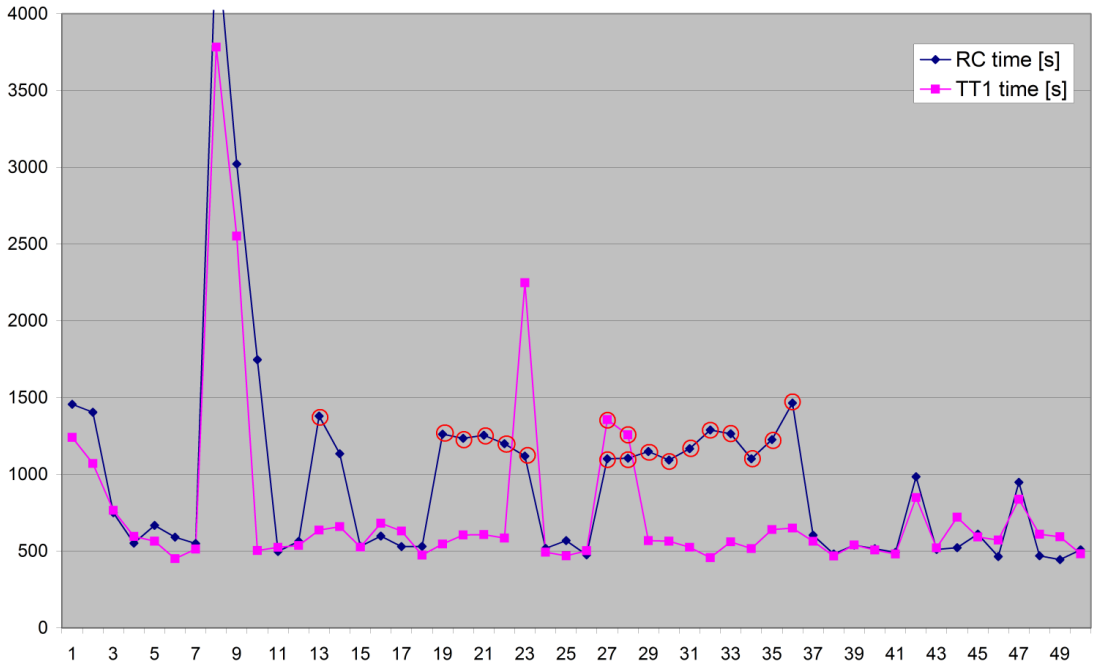

Figure 5.8: Workflow execution times in the third test series.

	RC	TT1
sum of execution times [s]	48725	38197
arithmetic average [s]	974,51	763,93
standard deviation	699,72	594,91

Table 5.3: Evaluation of execution times in series 3.

not considered.

Fig. 5.8 shows the workflow execution times in a diagram. Workflows with timeout are denoted by red circles. The numerical results are given in table 5.3. During series 3, much more peaks in the waiting time occurred than in series 1 because MediGRID utilization was much higher during this test period. That preferred the RC instance which can submit the first jobs before TT1 and sometimes led to better results for RC. Nevertheless, series 3 shows a clear qualitative and quantitative improvement for the TT1 instance with combined quality q''. While RC had timeouts in 16 out of 50 test runs, only 2 timeouts happened in TT1. The measured speedup factor of TT1 over RC is 1.28. It would have been much higher even if no timeout for the Grid jobs had been set.

5.6 Summary Discussion

Traditional workflow scheduling strategies that perform full-ahead scheduling of the complete workflow graph using heuristics such as list scheduling or genetic algorithms have problems to cope with the dynamic and unpredictable nature of Grid resources. Therefore, they typically require frequent rescheduling during the execution of the workflow. Dynamic Grid schedulers can deliver good performance in the mapping of single tasks to resources if they are able to make accurate evaluations of resource characteristics. An important quantity here is the estimated input-queue waiting-time of the clusters at Grid sites. However, single task schedulers inherently lack performance in case of complex, unbalanced application workflows.

The two-tier approach for Grid workflow scheduling is a hybrid scheduling model that combines the advantages of full-ahead and just-in-time scheduling. It employs the weighting and ranking phase of the HEFT algorithm to optimize the execution time of the overall workflow. Instead of the static mapping phase of HEFT the second tier dynamically assigns tasks to resources based on the predicted queue waiting time. This procedure is distinct from the scheduling approaches found in related general-purpose Grid workflow systems. In these systems, a hybrid scheduling process is realized either by partitioning the workflow and performing full-ahead scheduling dynamically on the subworkflows, or by rescheduling the workflow. The rescheduling is triggered in case of unsuccessful in task execution or when the state of the Grid changes. The two-tier approach does not depend on rescheduling, however tasks are resubmitted to alternate machines by GWES's integrated fault management in case of unsuccessful execution.

The two-tier approach replaces the previous scheduler of GWES that only considers the mapping of individual Petri net transitions. The new workflow-level scheduling-phase utilizes historical information that was gained during previous executions of applications to minimize the execution time of the workflow. To apply the HEFT algorithm to Petri nets, some modifications of the algorithm were necessary that handle non-DAG characteristics of Petri nets such as control flow transitions and loops.

The grid-level scheduling-phase is an improved version of the previous just-in-time scheduling of GWES. By this means, the two-tier approach ensures the capability

to correctly handle all Petri-net control-flow constructs as the previous implementation. Furthermore, this makes our new approach downward compatible to the existing workflow engine of GWES. If necessary, the grid-level scheduling could also be employed stand-alone without the workflow-level scheduling.

The second tier extends the previous scheduler of GWES with support of multiple scheduling criteria. Besides the existing Grid-oriented optimization of load-balancing between resources, it allows for additional prioritization on workflow and user level. The resource selection is greatly improved by the incorporation of our methodology to predict the input-queue waiting-times of the prevalent cluster resources. To align queueing delay with other performance parameters, we extended the previous quality metric for clusters by an additional formula that translates queue waiting time into quality values. Table 5.4 compares the previous scheduler of GWES with the two-tier approach using the taxonomies from [87] as in Section 3.2.

For the evaluation of the two-tier approach both tiers were implemented and integrated into GWES. The subsequent assessment confirmed the effectiveness of combining full-ahead and just-in-time scheduling into a hybrid approach. Additionally, the results showed that input-queue waiting-time prediction is a requirement for efficient workflow execution in production Grids. It not only minimizes the turnaround time of Grid jobs but also provides reliable job execution by recognizing peaks in waiting time that could otherwise cause the abortion of application execution.

The two-tier approach addresses the competing-schedulers problem as well as the lack-of-control problem 4.2 by means of the dynamic resource selection in tier 2. The just-in-time scheduling postpones the mapping of tasks to the latest possible point in time. This way, state changes on Grid resources that are caused by competing schedulers can be recognized and reacted on during resource selection.

In general, the competing-schedulers problem cannot be solved by a single meta-scheduler alone. It rather has to be considered in the design of Grids, e.g. by appointing dedicated resources for the VOs. Provided such exclusive resource access, meta-schedulers can perform full planning of execution times and circumvent delays. In D-Grid, the problem is recently approached by the DGSI project [124] that aims at introducing negotiation capabilities into the participating Grid schedulers. As a result of these negotiations, schedulers delegate jobs or resources

		previous scheduler	two-tier scheduler
Workflow model	task oriented	+	+
	task and data oriented	-	-
	DAG	-	-
	extended digraph	+	+
	simplified DAG	-	-
	tunable workflows	-	-
	single input workflows	+	+
	pipelined workflows	-	-
Criteria	multiple criteria	(+)	+
	execution time	-	+
	execution cost	-	-
	reliability	-	-
	data quality	-	-
	generic criteria model	-	-
	Grid-oriented criteria	+	+
	adaptive cost model	-	-
Sched. process	multiple workflows	+	+
	just-in-time	+	-
	full-ahead	-	-
	hybrid	-	+
	advance reservation	-	-
Res.	heterogeneous resources	+	+
	multiprogrammed resources	+	+
Task	moldable tasks	-	-
	malleable tasks	-	-
	migrative tasks	-	-

Table 5.4: Comparison of scheduling methods.

among each other and use resources exclusively.

Advance reservations are a technical means to gain exclusive access to a set of resources. They address all of the discovered problems, especially the non-continuously differentiable-function problem. Therefore, reservations are considered by several research groups. However, in the production environment reservation capabilities are not well supported by the LRMS and remain a challenge.

After the performance assessment, the two-tier approach has been integrated into the official GWES release. It is deployed on the MediGRID portal server and used for the scheduling in MediGRID. Similarly, it can be employed in other current and future projects that also build upon GWES as workflow system.

Chapter 6

Conclusion

In this thesis, we have shown the development process of gridifying an application for the MediGRID virtual organization by means of a case study. The process is subdivided into steps and based on the middleware that is employed such as the Globus Toolkit, the GWES workflow system, and the Grid portal framework. The central step is to divide the application scenario into a number of tasks which are represented as a Petri net workflow. The described gridification process can be applied to an existing application without changes to the source code. However, it should be done by the application developers since it requires insight into the application's mode of operation. Furthermore, Grid coaching for the application developers is necessary. This role can be fulfilled by Grid experts at the computing centers. As with traditional parallelization, they can help with training on the new middleware layers imposed by the Grid. Additionally, computing centers must install the application and middleware software components.

The benefit of this effort is that in the end, the applications are accessible according to the "software as a service" model. The portlets offer an user interface that corresponds to the application domain. The end users only have to register at the Grid portal. Apart from that, they need no detailed Grid knowledge to profit from the wide range of computing resources that allows tackling larger problem sizes. For computing centers, Grid technology offers another mainstay besides traditional parallel computing. With Grids they can offer computing services to new customer groups.

The first results have shown that Grid technology can help to reduce the execution

time of the gene prediction program AUGUSTUS, which is exemplary for other biomedical applications. In particular, when a project requires several application scenarios with different parameter configurations, Grid technology can accelerate the overall project progress considerably. In addition, accessibility and usability can be achieved by Grid portals.

The software components used in MediGRID are specific choices from a range of existing middleware solutions. They are accepted among the Grid community and have proven to be appropriate. The Globus Toolkit version 4 consists of several components, out of which some need to be extended for custom application scenarios. The Globus services employed in MediGRID belong to core components and are reliable for production use. The MediGRID portal also builds on a widely deployed solution, therefore further development concentrates on the application specific portlets. The GWES workflow system was adopted for many projects and is in use in several virtual organizations. However, it is continuously enhanced to improve workflow execution.

It was our aim to design a meta-scheduler for Petri net workflows that offers efficient and reliable job execution in production environments. To this end, we performed a runtime analysis of the time-dependant availability of D-Grid resources. A measurement program was installed at 11 large clusters each located at a different computing center. As a result of our measurements in D-Grid, we have discovered four intrinsic problems that limit the possibilities of meta-scheduling. These problems are: competing-schedulers, lack-of-control, information-insufficiency, and the non-continuously differentiable-function problem. Grid scheduling in D-Grid is a difficult task, because Grid schedulers only have the role of "power users" competing with other schedulers. They have no direct control over the LRMS, and the information available to base the scheduling decisions upon is very limited. Finally, many resources are highly utilized so that queue waiting times can last up to hours.

To improve the selection of Grid resources, we developed and evaluated a methodology to predict the queue waiting times of cluster resources. We found out that three different scenarios could be identified for which three disjoint estimation methods could be given that predict the respective situation best. Furthermore, three selection algorithms were proposed to automatically select the best estimation method due to the current situation on the site. The evaluation showed that

the selection algorithm that uses 18 states and 6 accuracy levels for every estimation method performs best. With that selection algorithm prediction works very well on most of the examined D-Grid sites and helps to recognize the peaks in waiting times which can last up to hours.

For the scheduling of workflow tasks we presented the two-tier approach that combines contemporary scheduling strategies for workflows and for Grids, and that utilizes historical information from previous executions of applications as well as estimations of pre-execution delay. The first tier performs a workflow-level scheduling that employs parts of the "Heterogeneous Earliest-Finish-Time" algorithm and execution time predictions to create a full-ahead static schedule of tasks. This improves the scheduling performance in case of complex, unbalanced application workflows. The second tier performs a Grid-level scheduling by means of dynamic just-in-time mapping of the prioritized tasks to the Grid resources. It considers fairness between users, load distribution among the sites, and performance metrics that incorporate the predicted queue waiting times.

We have integrated the two-tier approach into the GWES workflow system where it replaces the previous just-in-time scheduler. It is compatible to MediGRID workflows because the second tier keeps the capability to handle all Petri net constructs like the previous implementation. Afterwards, the new approach was assessed by many simultaneous runs of a reference workflow. To this end, a testbed that reproduces the MediGRID production system was set up to directly compare the previous version of the MediGRID scheduler with one version that used only the new Grid-level scheduling, and a second version that used Grid- and workflow-level scheduling. The results showed that both tiers decrease the average workflow execution times. The performance benefit was up to 28%. The first tier also decreases execution time in case of no waiting times. The second tier is especially effective in case of high resource utilization. To achieve this, two metrics for the potential availability of cluster resources are employed. By a combination of both metrics, the best results were achieved, resulting in an efficient and very robust scheduler. The two-tier scheduler is now in production use in the MediGRID virtual organization.

Bibliography

[1] A. S. Tanenbaum and M. van Steen, *Verteilte Systeme. Prinzipien und Paradigmen*. Pearson Studium, 2nd ed., 2008.

[2] P. Rechenberg and G. Pomberger, *Informatik-Handbuch*. Carl Hanser Verlag, 3rd ed., 2002.

[3] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss, "Overview of the I-WAY: Wide-area visual supercomputing," *International Journal of High Performance Computing Applications*, vol. 10, no. 2-3, p. 123, 1996.

[4] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[5] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of High Performance Computing Applications*, vol. 15, pp. 200–222, August 2001.

[6] I. Foster, "Introduction to the Grid, COMDEX 2003." http://www.mcs.anl.gov/~itf/Talks/Comdex%20Grid%20Foster%20November%202003.ppt, 18 November 2003. [Online; accessed 1-August-2011].

[7] I. Foster, "What is the Grid? A Three Point Checklist," *GRID today*, vol. 1, no. 6, 2002.

[8] I. Foster, "The Grid: A New Infrastructure for 21st Century Science," *Physics Today*, vol. 55, no. 2, pp. 42–47, 2002.

[9] I. Foster, "Globus Toolkit 4, Tutorial, 1st Intl. Conf. on e-Science and Grid Computing." http://www.mcs.anl.gov/~itf/Talks/051205%20Globus%20

Tutorial%20eScience.ppt, 12 December 2005. [Online; accessed 1-August-2011].

[10] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," in *Open Grid Service Infrastructure WG, Global Grid Forum*, (Edinburgh), 22 June 2002.

[11] World Wide Web Consortium (W3C), "Web Services Description Language (WSDL) 1.1." http://www.w3.org/TR/2001/NOTE-wsdl-20010315, 2001. [Online; accessed 28-May-2012].

[12] "MediGRID." http://www.medigrid.de. [Online; accessed 10-June-2011].

[13] H. Neuroth, M. Kerzel, and W. Gentzsch, eds., *German Grid Initiative D-Grid*. Universitätsverlag Göttingen, 2007.

[14] D. Krefting, J. Bart, K. Beronov, O. Dzhimova, J. Falkner, M. Hartung, A. Hoheisel, T. A. Knoch, T. Lingner, Y. Mohammed, K. Peter, E. Rahm, U. Sax, D. Sommerfeld, T. Steinke, T. Tolxdorff, M. Vossberg, F. Viezens, and A. Weisbecker, "MediGRID: Towards a user friendly secured grid infrastructure," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 326 – 336, 2009.

[15] "Services@MediGRID." http://services.medigrid.de. [Online; accessed 10-June-2011].

[16] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *NPC* (H. Jin, D. A. Reed, and W. Jiang, eds.), vol. 3779 of *Lecture Notes in Computer Science*, pp. 2–13, Springer, 2005.

[17] A. Hoheisel, "Grid Workflow Execution Service - Dynamic and interactive execution and visualization of distributed workflows," in *Proceedings of the Cracow Grid Workshop*, vol. 2, pp. 13–24, 2006.

[18] F. Neubauer, A. Hoheisel, and J. Geiler, "Workflow-based grid applications," *Future Gener. Comput. Syst.*, vol. 22, no. 1, pp. 6–15, 2006.

[19] M. Stanke, O. Schoeffmann, B. Morgenstern, and S. Waack, "Gene prediction in eukaryotes with a generalized hidden Markov model that uses hints from external sources," *BMC Bioinformatics*, vol. 7, p. 62, 2006.

[20] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling forheterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[21] D. Sommerfeld, T. Lingner, and H. Richter, "Stepwise Enabling of AUGUSTUS for MediGRID," Tech. Rep. IfI-07-12, Clausthal University of Technology, 2007.

[22] D. Sommerfeld, T. Lingner, M. Stanke, B. Morgenstern, and H. Richter, "AUGUSTUS at MediGRID: Adaption of a bioinformatics application to grid computing for efficient genome analysis," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 337 – 345, 2009.

[23] D. Sommerfeld and H. Richter, "A Novel Approach to Workflow Scheduling in MediGRID," Tech. Rep. IfI-09-07, Clausthal University of Technology, 2009.

[24] D. Sommerfeld and H. Richter, "A two-tier approach to efficient workflow scheduling in MediGRID," in *Grid-Technologie in Göttingen - Beiträge zum Grid-Ressourcen-Zentrum GoeGrid* (U. Schwardmann, ed.), vol. GWDG-Bericht Nr. 74, pp. 39–51, Göttingen, Germany: Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, 2009.

[25] D. Sommerfeld and H. Richter, "Problems and Approaches of Workflow Scheduling in MediGRID," in *2009 Fifth IEEE International Conference on e-Science*, (Oxford, United Kingdom), pp. 223–230, 9–11 December 2009.

[26] D. Sommerfeld and H. Richter, "A Two-Tier Approach to Grid Workflow Scheduling," in *2009 2nd IEEE International Conference on Computer Science and its Applications*, (Jeju, Korea (South)), pp. 267–273, 10–12 December 2009.

[27] D. Sommerfeld and H. Richter, "Efficient Grid Workflow Scheduling Using a Two-Tier Approach," in *Proceedings of the Ninth HealthGrid conference 2011*, (Bristol, United Kingdom), 27–28 June 2011.

[28] E. Ghedin, S. Wang, D. Spiro, E. Caler, Q. Zhao, J. Crabtree, J. E. Allen, A. L. Delcher, D. B. Guiliano, D. Miranda-Saavedra, S. V. Angiuoli, T. Creasy, P. Amedeo, B. Haas, N. M. El-Sayed, J. R. Wortman, T. Feldblyum, L. Tallon, M. Schatz, M. Shumway, H. Koo, S. L. Salzberg, S. Schobel, M. Pertea, M. Pop, O. White, G. J. Barton, C. K. S. Carlow, M. J. Crawford, J. Daub, M. W. Dimmic, C. F. Estes, J. M. Foster, M. Ganatra, W. F. Gregory, N. M. Johnson, J. Jin, R. Komuniecki, I. Korf, S. Kumar, S. Laney, B.-W. Li, W. Li, T. H. Lindblom, S. Lustigman, D. Ma, C. V. Maina, D. M. A. Martin, J. P. McCarter, L. McReynolds, M. Mitreva, T. B. Nutman, J. Parkinson, J. M. Peregrin-Alvarez, C. Poole, Q. Ren, L. Saunders, A. E. Sluder, K. Smith, M. Stanke, T. R. Unnasch, J. Ware, A. D. Wei, G. Weil, D. J. Williams, Y. Zhang, S. A. Williams, C. Fraser-Liggett, B. Slatko, M. L. Blaxter, and A. L. Scott, "Draft genome of the filarial nematode parasite Brugia malayi," *Science*, vol. 317, pp. 1756–1760, Sep 2007. Comparative Study.

[29] V. Nene, J. R. Wortman, D. Lawson, B. Haas, C. Kodira, Z. J. Tu, B. Loftus, Z. Xi, K. Megy, M. Grabherr, Q. Ren, E. M. Zdobnov, N. F. Lobo, K. S. Campbell, S. E. Brown, M. F. Bonaldo, J. Zhu, S. P. Sinkins, D. G. Hogenkamp, P. Amedeo, P. Arensburger, P. W. Atkinson, S. Bidwell, J. Biedler, E. Birney, R. V. Bruggner, J. Costas, M. R. Coy, J. Crabtree, M. Crawford, B. Debruyn, D. Decaprio, K. Eiglmeier, E. Eisenstadt, H. El-Dorry, W. M. Gelbart, S. L. Gomes, M. Hammond, L. I. Hannick, J. R. Hogan, M. H. Holmes, D. Jaffe, J. S. Johnston, R. C. Kennedy, H. Koo, S. Kravitz, E. V. Kriventseva, D. Kulp, K. Labutti, E. Lee, S. Li, D. D. Lovin, C. Mao, E. Mauceli, C. F. M. Menck, J. R. Miller, P. Montgomery, A. Mori, A. L. Nascimento, H. F. Naveira, C. Nusbaum, S. O'leary, J. Orvis, M. Pertea, H. Quesneville, K. R. Reidenbach, Y.-H. Rogers, C. W. Roth, J. R. Schneider, M. Schatz, M. Shumway, M. Stanke, E. O. Stinson, J. M. C. Tubio, J. P. Vanzee, S. Verjovski-Almeida, D. Werner, O. White, S. Wyder, Q. Zeng, Q. Zhao, Y. Zhao, C. A. Hill, A. S. Raikhel, M. B. Soares, D. L. Knudson, N. H. Lee, J. Galagan, S. L. Salzberg, I. T. Paulsen, G. Dimopoulos, F. H. Collins, B. Birren, C. M. Fraser-Liggett, and D. W. Severson, "Genome sequence of Aedes aegypti, a major arbovirus vector," *Science*, vol. 316, pp. 1718–1723, Jun 2007.

[30] M. Stanke, R. Steinkamp, S. Waack, and B. Morgenstern, "AUGUSTUS: a web server for gene finding in eukaryotes," *Nucleic Acids Res.*, vol. 32, pp. W309–312, Jul 2004.

[31] A. Vinogradov, "Genome size and chromatin condensation in vertebrates," *Chromosoma*, vol. 113, no. 7, pp. 362–369, 2005.

[32] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, pp. 403–410, Oct 1990.

[33] B. Morgenstern, "A space-efficient algorithm for aligning large genomic sequences," *Bioinformatics*, vol. 16, pp. 948–949, Oct 2000.

[34] B. Morgenstern, S. J. Prohaska, D. Pohler, and P. F. Stadler, "Multiple sequence alignment with user-defined anchor points," *Algorithms Mol Biol*, vol. 1, no. 1, p. 6, 2006.

[35] M. Stanke, A. Tzvetkova, and B. Morgenstern, "AUGUSTUS at EGASP: using EST, protein and genomic alignments for improved gene prediction in the human genome," *Genome Biol*, vol. 7 Suppl 1, pp. 1–8, 2006. Evaluation Studies.

[36] W. R. Pearson, "Rapid and sensitive sequence comparison with FASTP and FASTA.," *Methods Enzymol*, vol. 183, pp. 63–98, 1990.

[37] L. D. Stein, C. Mungall, S. Shu, M. Caudy, M. Mangone, A. Day, E. Nickerson, J. E. Stajich, T. W. Harris, A. Arva, and S. Lewis, "The generic genome browser: a building block for a model organism system database.," *Genome Res*, vol. 12, pp. 1599–1610, October 2002.

[38] I. Foster, K. Czajkowski, D. E. Ferguson, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke, "Modeling and managing state in distributed systems: the role of OGSI and WSRF," *Proceedings of the IEEE*, vol. 93, no. 3, pp. 604–612, 2005.

[39] The Globus Alliance, "Globus Toolkit 4.0 Documentation Overview." http://www.globus.org/toolkit/docs/4.0/GT4figure.jpg, 2011. [Online; accessed 1-August-2011].

[40] P. Tremblett, "X.509 certificates," *Dr. Dobb's Journal*, vol. 24, no. 7, pp. 42–51, 1999.

[41] Y. Demchenko, C. de Laat, and V. Ciaschini, "VO-based dynamic security associations in collaborative grid environment," in *CTS '06: Proceedings of the International Symposium on Collaborative Technologies and Systems*, (Washington, DC, USA), pp. 38–47, IEEE Computer Society, 2006.

[42] T. Dierks and C. Allen, "The TLS protocol," tech. rep., Certicom Network Working Group, 1999.

[43] J. Clark *et al.*, "XSL transformations (XSLT) version 1.0," *W3C recommendation*, vol. 16, no. 11, 1999.

[44] The Globus Alliance, "Reliable File Transfer Service (RFT)." http://www.globus.org/toolkit/docs/4.0/data/rft/. [Online; accessed 10-June-2011].

[45] The PostgreSQL Global Development Group, "PostgreSQL: The world's most advanced open source database." http://www.postgresql.org/, 2012. [Online; accessed 2-June-2012].

[46] The Globus Alliance, "Web Service Grid Resource Allocation and Management (WS GRAM)." http://www.globus.org/toolkit/docs/4.0/execution/wsgram/. [Online; accessed 10-June-2011].

[47] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," in *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, (London, UK), pp. 62–82, Springer-Verlag, 1998.

[48] The Globus Alliance, "GT 4.0 WS GRAM: Job Description Schema Doc ." http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html. [Online; accessed 10-June-2011].

[49] D. Krefting, S. Canisius, A. Hoheisel, H. Loose, T. Tolxdorff, and T. Penzel, "Grid based sleep research – Analysis of polysomnographies using a

grid infrastructure," *Future Generation Computer Systems*, vol. In Press, Corrected Proof, 2010.

[50] C. Boehme, T. Ehlers, J. Engelhardt, A. Félix, O. Haan, T. Kálmán, B. Neumair, U. Schwardmann, and D. Sommerfeld, "Instant-Grid: Fully automated middleware-deployment using a live-CD," in *ICNS '06: Proceedings of the International conference on Networking and Services*, p. 70, IEEE Computer Society, 2006.

[51] "BauVOGrid." http://www.bauvogrid.de. [Online; accessed 10-June-2011].

[52] "K-Wf Grid." http://www.kwfgrid.eu. [Online; accessed 10-June-2011].

[53] "CoreGRID - Network of Excellence." http://www.coregrid.net. [Online; accessed 10-June-2011].

[54] A. Hoheisel and H. Rose, "Konzept für das Scheduling von Workflow-Aktivitäten in Instant Grid," tech. rep., Fraunhofer Institut für Rechnerarchitektur und Softwaretechnik, June 2006.

[55] A. Wolf, "Spezifikation der D-Grid-Ressourcenbeschreibungssprache D-GRDL," DGI FG 2-4 Technical Report, Fraunhofer FIRST, 2007. [Online; accessed 10-June-2011].

[56] A. Hoheisel and M. Alt, "Petri nets," in *Workflows for eScience* (I. J. Taylor, D. Gannon, E. Deelman, and M. S. Shields, eds.), Springer-Verlag, 2006.

[57] M. Alt, A. Hoheisel, H.-W. Pohl, and S. Gorlatch, "A grid workflow language using high-level petri nets," in *PPAM2005* (R. W. et al., ed.), vol. 3911 of *LNCS*, pp. 715–722, Springer-Verlag Berlin Heidelberg, 2006.

[58] J. Novotny, M. Russell, and O. Wehrens, "Gridsphere: a portal framework for building collaborations," *Concurrency And Computation-Practice & Experience*, vol. 16, no. 5, pp. 503 – 513, 2004.

[59] O. Diaz and J. J. Rodriguez, "Portlets as web components: an introduction," *Journal of Universal Computer Science*, vol. 10, no. 4, pp. 454–472, 2004.

[60] Sun Developer Network, "JavaServer Pages Technology - Documentation." http://java.sun.com/products/jsp/docs.html, 2012. [Online; accessed 2-June-2012].

[61] StringTemplate.org, "StringTemplate Template Engine." http://www.stringtemplate.org/, 2012. [Online; accessed 2-June-2012].

[62] A. B. Downey, "Predicting queue times on space-sharing parallel computers," in *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pp. 209–218, IEEE, 1997.

[63] A. B. Downey, "Using queue time predictions for processor allocation," in *Job Scheduling Strategies for Parallel Processing*, pp. 35–57, Springer, 1997.

[64] D. Feitelson and L. Rudolph, "Toward convergence in job schedulers for parallel supercomputers," in *Job Scheduling Strategies for Parallel Processing*, pp. 1–26, Springer, 1996.

[65] W. Smith, V. Taylor, and I. Foster, "Using run-time predictions to estimate queue wait times and improve scheduler performance," in *Job Scheduling Strategies for Parallel Processing*, pp. 202–219, Springer, 1999.

[66] D. Jackson, Q. Snell, and M. Clement, "Core algorithms of the maui scheduler," in *Job Scheduling Strategies for Parallel Processing*, pp. 87–102, Springer, 2001.

[67] W. Smith, "Improving resource selection and scheduling using predictions," in *Grid resource management*, pp. 237–253, Kluwer Academic Publishers, 2004.

[68] H. Li, D. Groep, J. Templon, and L. Wolters, "Predicting job start times on clusters," in *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pp. 301–308, IEEE, 2004.

[69] J. Brevik, D. Nurmi, and R. Wolski, "Predicting bounds on queuing delay in space-shared computing environments," in *Workload Characterization, 2006 IEEE International Symposium on*, pp. 213–224, IEEE, 2006.

[70] J. Brevik, D. Nurmi, and R. Wolski, "Predicting bounds on queuing delay for batch-scheduled parallel machines," in *Proceedings of the eleventh ACM*

SIGPLAN symposium on Principles and practice of parallel programming, pp. 110–118, ACM, 2006.

[71] D. Nurmi, A. Mandal, J. Brevik, C. Koelbel, R. Wolski, and K. Kennedy, "Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, p. 119, ACM, 2006.

[72] D. Nurmi, J. Brevik, and R. Wolski, "Qbets: Queue bounds estimation from time series," in *Job Scheduling Strategies for Parallel Processing*, pp. 76–101, Springer, 2008.

[73] D. Chaturvedi, S. Premdayal, and A. Chandiok, "Short-term load forecasting using soft computing techniques," *International Journal of Communications, Network and System Sciences*, vol. 3, no. 1, pp. 270–279, 2010.

[74] C. Catlett, "The philosophy of teragrid: building an open, extensible, distributed terascale facility," in *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*, pp. 8–8, IEEE, 2002.

[75] O. Sonmez, N. Yigitbasi, A. Iosup, and D. Epema, "Trace-based evaluation of job runtime and queue wait time predictions in grids," in *Proceedings of the 18th ACM international symposium on High performance distributed computing*, pp. 111–120, ACM, 2009.

[76] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. Epema, "The grid workloads archive," *Future Generation Computer Systems*, vol. 24, no. 7, pp. 672–686, 2008.

[77] R. Wolski, N. Spring, and J. Hayes, "The network weather service: a distributed resource performance forecasting service for metacomputing," *Future Generation Computer Systems*, vol. 15, no. 5, pp. 757–768, 1999.

[78] R. Wolski, L. Miller, G. Obertelli, and M. Swany, "Performance information services for computational grids," *International Series In Operations Research And Management Science*, pp. 193–214, 2003.

[79] J. Nabrzyski, J. M. Schopf, and J. Weglarz, eds., *Grid resource management: state of the art and future trends*. Norwell, MA, USA: Kluwer Academic Publishers, 2004.

[80] J. Schopf, "Ten actions when grid scheduling: the user as a grid scheduler," *International series in Operations Research and Management Science*, 2003.

[81] F. Dong and S. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," *School of Computing, Queen's University, Kingston, Ontario*, 2006.

[82] E. Huedo, R. Montero, and I. Llorente, "The GridWay framework for adaptive scheduling and execution on grids," *SCPE*, vol. 2006, p. 2007, 2004.

[83] GridWay Project Leads, "GridWay Scheduling Capabilities." http://www.gridway.org/doku.php?id=about:functionality:schedcapab, 2012. [Online; accessed 2-June-2012].

[84] J. Frey, "Condor dagman: Handling inter-job dependencies," *University of Wisconsin, Dept. of Computer Science, Tech. Rep*, 2002.

[85] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing*, vol. 3, no. 3, pp. 171–200, 2005.

[86] M. Wieczorek, A. Hoheisel, and R. Prodan, *Grid middleware and services: challenges and solutions*, ch. Taxonomies of the Multi-criteria Grid Workflow Scheduling Problem, pp. 237–264. CoreGRID, Springer, June 2008.

[87] M. Wieczorek, A. Hoheisel, and R. Prodan, "Towards a general model of the multi-criteria workflow scheduling on the grid," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 237–256, 2009.

[88] D. Abramson, R. Buyya, and J. Giddy, "A computational economy for grid computing and its implementation in the nimrod-g resource broker," *Future Generation Computer Systems*, vol. 18, no. 8, pp. 1061–1074, 2002.

[89] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, *et al.*, "New grid scheduling and rescheduling methods in the grads project," *International Journal of Parallel Programming*, vol. 33, no. 2, pp. 209–229, 2005.

[90] H. Dail, O. Sievert, F. Berman, H. Casanova, A. YarKhan, S. Vadhiyar, J. Dongarra, C. Liu, L. Yang, D. Angulo, *et al.*, "Scheduling in the grid application development software project," *International Series In Operations Research And Management Science*, pp. 73–98, 2003.

[91] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt, "Qos support for time-critical grid workflow applications," in *Proceedings of the First International Conference on e-Science and Grid Computing*, pp. 108–115, IEEE Computer Society, 2005.

[92] I. Brandic, S. Pllana, and S. Benkner, "Amadeus: A holistic service-oriented environment for grid workflows," in *Grid and Cooperative Computing Workshops, 2006. GCCW'06. Fifth International Conference on*, pp. 259–266, IEEE, 2006.

[93] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman, "Workflow management in griphyn," in *Grid resource management*, pp. 99–116, Kluwer Academic Publishers, 2004.

[94] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[95] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H. Truong, *et al.*, "Askalon: A development and grid computing environment for scientific workflows," *Workflows for e-Science*, pp. 450–471, 2007.

[96] M. Mair, J. Qin, M. Wieczorek, and T. Fahringer, "Workflow conversion and processing in the askalon grid environment," in *2nd Austrian Grid Symposium*, pp. 67–80, 2007.

[97] M. Wieczorek, M. Siddiqui, A. Villazon, R. Prodan, and T. Fahringer, "Applying advance reservation to increase predictability of workflow execution on the grid," in *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*, pp. 82–82, IEEE, 2006.

[98] T. Fahringer, J. Qin, and S. Hainzer, "Specification of grid workflow applications with agwl: an abstract grid workflow language," in *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, vol. 2, pp. 676–685, IEEE, 2005.

[99] A. Hoheisel, "User tools and languages for graph-based grid workflows," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1101–1113, 2006.

[100] L. Dutka, J. Kitowski, and N. S., "Automatic application builder – Tool for automatic service selection," in *Proceedings of Cracow'06 Grid Workshop*, pp. 31–38, ACC Cyfronet AGH, 2006.

[101] A. Kertesz, G. Sipos, and P. Kacsuk, "Multi-grid brokering with the p-grade portal," in *2nd Austrian Grid Symposium*, pp. 166–178, OCG Verlag, 2006.

[102] B. Schuller, B. Demuth, H. Mix, K. Rasch, M. Romberg, S. Sild, U. Maran, P. Bała, E. Del Grosso, M. Casalegno, N. Piclin, M. Pintore, W. Sudholt, and K. Baldridge, "Chemomentum – unicore 6 based infrastructure for complex applications in science and technology," in *UNICORE Summit 2007*, pp. 82–93, Springer-Verlag, 2007.

[103] "The Knowledge Grid Project." http://grid.deis.unical.it/kgrid/. [Online; accessed 2-June-2012].

[104] R. Buyya, *Economic-based distributed resource management and scheduling for grid computing*. PhD thesis, Monash University, 2002.

[105] Z. Yu and W. Shi, "An adaptive rescheduling strategy for grid workflow applications," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–8, IEEE, 2007.

[106] J. Qin, M. Wieczorek, K. Plankensteiner, and T. Fahringer, "Towards a light-weight workflow engine in the askalon grid environment," in *Towards next generation grids: proceedings of the CoreGRID Symposium 2007, August 27-28, Rennes, France*, p. 239, Springer-Verlag New York Inc, 2007.

[107] R. Yahyapour, *Design and evaluation of job scheduling strategies for grid computing*. PhD thesis, Universität Dortmund, 2002.

[108] A. Streit, *Self-tuning job scheduling strategies for the resource management of HPC systems and computational grids*. PhD thesis, Universität Paderborn, 2003.

[109] C. Schenk and U. Tietze, *Halbleiter-schaltungstechnik*. Springer, 1999.

[110] D. Gross, J. Shortle, J. Thompson, and C. Harris, *Fundamentals of queueing theory*. John Wiley & Sons, 2008.

[111] N. Morrison, *Introduction to Fourier analysis*. Wiley New York, 1994.

[112] T. Butz, *Fourier transformation for pedestrians*. Springer, 2006.

[113] P. T. Debevec, "The Excel FFT Function v1.2." http://online.physics.uiuc.edu/courses/phys401/spring09/Lectures/Excel/The_Excel_FFT_Function_v1.2.pdf, 2008. [Online; accessed 10-June-2011].

[114] L. Klingenberg, "Frequency Domain Using Excel." http://online.sfsu.edu/~larryk/Common%20Files/Excel.FFT.pdf, 2005. [Online; accessed 10-June-2011].

[115] W. Schiffmann, R. Schmitz, and J. Weiland, *Technische Informatik*. Springer, 2001.

[116] M. Wieczorek, R. Prodan, and T. Fahringer, "Comparison of workflow scheduling strategies on the grid," *Lecture Notes In Computer Science*, vol. 3911, pp. 792–800, 2006.

[117] H. Zhao and R. Sakellariou, "An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm," *Euro-Par 2003 Parallel Processing*, pp. 189–194, 2003.

[118] R. Sakellariou and H. Zhao, "A hybrid heuristic for dag scheduling on heterogeneous systems," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pp. 111–123, IEEE, 2004.

[119] Standard Performance Evaluation Corporation, "SPEC CPU2006 Results." http://www.spec.org/cpu2006/results/, 2011. [Online; accessed 1-August-2011].

[120] E. de Oliveira, "Jawari - A grid benchmarking service," in *Proceedings of the German e-Science Conference*, (Baden-Baden, Germany), 2–4 May 2007.

[121] Fraunhofer FIRST, "Generic Workflow Execution Service." http://www.gridworkflow.org/kwfgrid/gwes-web/, 2011. [Online; accessed 1-June-2011].

[122] The Apache Software Foundation, "Apache Maven Project." http://maven.apache.org/, 2011. [Online; accessed 1-August-2011].

[123] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[124] G. Birkenheuer, A. Brinkmann, M. Högqvist, A. Papaspyrou, B. Schott, D. Sommerfeld, and W. Ziegler, "Infrastructure federation through virtualized delegation of resources and services," *Journal of Grid Computing*, vol. 9, pp. 355–377, 2011.

List of Figures

List of Tables